OULUN AMMATTIKORKEAKOULU

Veli-Pekka Mäki

RADIOVERKKOJÄRJESTELMIEN TESTAUSDATAN KÄSITTELY JA VISUALISOINTI AVOIMEN LÄHDEKOODIN BIG DATA -TYÖKALUILLA

RADIOVERKKOJÄRJESTELMIEN TESTAUSDATAN KÄSITTELY JA VISUALISOINTI AVOIMEN LÄHDEKOODIN BIG DATA -TYÖKALUILLA

Veli-Pekka Mäki
Opinnäytetyö
Kevät 2018
Tietojenkäsittelyn koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma, Digitaaliset palvelut -suuntautumisvaihtoehto

Tekijä: Veli-Pekka Mäki
Opinnäytetyön nimi: Radioverkkojärjestelmien testausdatan käsittely ja visualisointi avoimen lähdekoodin Big Data -työkaluilla
Työn ohjaaja: Jukka Kaisto
Työn valmistumislukukausi ja -vuosi: Kevät 2018 Sivumäärä: 85 + 3

Opinnäytetyön toimeksiantaja on Nokia Solutions and Networks Oy. Opinnäytetyön tarkoituksena on testata kahta avoimen lähdekoodin hakukonetta: Elasticsearch ja Apache Solr ja selvittää, miten ne soveltuvat radioverkkojärjestelmien ohjelmistojen testausympäristön tuottaman lokitiedon käsittelyyn. Työn tavoitteena on selvittää käytännössä, miten testausdataa voidaan viedä hakukoneeseen, indeksoitua tietoa voidaan hakea ja esittää visuaalisesti.

Opinnäytetyön teoriaosuudessa käsitellään erilaisia lokitietojen tietomuotoja. Käsiteltäviä tietomuotoja ovat: XML, JSON, CSV ja Syslog. XML-tietomuodon yhteydessä käsitellään XML-dokumenttioliomallin lisäksi XML-tietomuodon käsittelykieliä: XPATH ja XSLT. Syslog-tietomuodon yhteydessä käsitellään Regex-lausekkeita (regular expression, suom. säännöllinen lauseke), joita käytetään merkkijonojen käsittelyssä. Seuraavaksi selvitetään hakukoneiden toimintaperiaatetta ja Lucene hakukonekirjastoa, johon molemmat käsiteltävät hakukoneet pohjautuvat. Teoriaosuuden lopussa käsitellään Big Data -tallennusratkaisua Hadoop ja sen HDFS-tiedostojärjelmää.

Opinnäytetyössä on käytännön esimerkit lokitiedon viemisestä hakukoneeseen, kun lokitieto on XML- ja CSV-tietomuotoa. Elasticsearch hakukoneeseen tieto viedään Logstash ohjelmalla, jonka jälkeen tietoa haetaan ja visualisoidaan Kibana ohjelmassa. Apache Solr hakukoneessa XML-tiedosto viedään hakukoneeseen käyttämällä XSLT-kieltä ja Data Import -pyyntökäsittelijää. Apache Solr hakukoneeseen vietyä tietoa visualisoidaan Banana ohjelmassa.

Opinnäytetyön perusteella Elasticsearch hakukone vaikuttaa toimivalta kokonaisuudelta, johon kuuluu Logstash ja Kibana (ELK stack). Kibana ohjelmalla on mahdollista saada aikaan suhteellisen vaivattomasti näyttäviä visualisointeja. Apache Solr hakukoneen yhteydessä ei tule visualisointi ohjelmaa vaan työssä käytetty Banana ohjelma on asennettava erikseen. Molemmille hakukoneille löytyy hyvät dokumentaatiosivut internetistä, mutta Elasticsearch hakukoneen käyttäjäyhteisö on suurempi ja internetistä löytyy runsaasti tietoa ohjelmaan liittyen. Molemmilla hakukoneilla esimerkeissä käytetyt lokitiedot saatiin vietyä hakukoneeseen, mutta Logstash ohjelman käytön Elasticsearch hakukoneen kanssa oppi nopeammin kuin Apache Solr hakukoneen indeksointitavat.

Hadoop tiedostojärjestelmää käytettäessä kannattaa tutustua Elasticsearch hakukoneen Hadoop-liitäntään: ES Hadoop, jolloin käytössä on Hadoop työkaluja, kuten Spark ja Pig. Myös Apache Solr hakukoneelle on tarjolla Hadoop-liitäntöjä, kuten HDFS for Solr, Pig for Solr ja Spark for Solr.

Asiasanat: big data, lokitiedosto, hakukone, visualisointi

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Business Information Systems,
Digital Services

Author: Veli-Pekka Mäki
Title of thesis: Using Open Source Big Data tools for analyzing and visualizing test data of radio network systems
Supervisor: Jukka Kaisto
Term and year when the thesis was submitted: Spring 2018 Number of pages: 85 + 3

The thesis was done for Nokia Solutions and Networks Oy. The purpose of the thesis is to test two open source search engines: Elasticsearch and Apache Solr, and find out how they are suitable for processing log data generated by the software testing environment of radio network systems. The aim of the thesis is to find out how the test data can be imported to the seach engine, the indexed data can be searched and displayed visually.

In the theoretical part of the thesis the following log data formats are handled: XML, JSON, CSV, and Syslog. Along with XML data format XPATH and XSLT processing languages are used, and with Syslog data format Regular Expressions are introduced how they can be used for defining string search patterns. Next, the search engine operating principle and the Lucene search engine library are presented which both tested search engines are based on. At the end of the theoretical section, the Big Data Storage Solution Hadoop and its HDFS file system is introduced.

The thesis includes practical examples of crawling the log information to the search engine when log data is an XML and CSV data format. To the Elasticsearch search engine, data is imported via Logstash software, after which information is searched and visualized in the Kibana software. To the Apache Solr search engine, the XML file is imported to the search engine by using the XSLT language and the Data Import request handler. The information imported to the Apache Solr search engine is visualized in the Banana software.

Based on the thesis, the Elasticsearch search engine affects to be working entity, including Logstash and Kiban (ELK stack). By using Kibana software it is possible to achieve relatively easily impressive visualizations. The Apache Solr search engine does not have a visualization software, but the Banana program must be installed separately. Both search engines have good documentation pages on the internet, but the Elasticsearch search engine's user community is bigger and due to this there is plenty of information available about the subject. The log files used in examples could be imported to the both search engines, but Logstash usage with the Elasticsearch search engine could be learned faster than the Apache Solr search engine crawling methods.

When using the Hadoop file system, you should familiarize yourself with the Hadoop interface of the Elasticsearch search engine: ES Hadoop, which makes it possible to use Hadoop tools such as Spark and Pig. The Apache Solr search engine also offers Hadoop interfaces such as: HDFS for Solr, Pig for Solr and Spark for Solr.

Keywords: big data, log file, search engine, visualization

SISÄLLYS

1 JOHDANTO

Olemme lisääntyvässä määrin riippuvaisia ohjelmistojen toimivuudesta, koska tänä päivänä niitä käytetään kaikkialla helpottamaan arkeamme. Ohjelmistot ja järjestelmät monimutkaistuvat ja kasvavat koko ajan ja siitä huolimatta niiden oletetaan toimivan virheettömästi koko ajan. Ohjelmistojen ja järjestelmien testaus on välttämätöntä, jotta ohjelmistovirheet löydettäisiin jo testausvaiheessa jolloin ne voidaan vielä korjata ennen tuotteen luovuttamista asiakkaan käyttöön. Näin toimimalla parannetaan myös asiakkaalle toimitetun tuotteen laatua. (Homes 2011, 1.)

Radioverkkojärjestelmien kuormitus-, suorituskyky- ja stabiilisuustestaus tuottaa suuren määrän dataa, josta on vaikeaa saada kokonaiskuvaa ilman tiedon jäsentelyä ja visualisointia. Tästä datasta tarvitaan erilaisia näkymiä eri organisaatioille sekä valikoitua tietoa tukiasematestaajien käyttöön.

Data-analysoinnin välittömänä tavoitteena on saada selvyys tukiasemaohjelmistojen päivitysten toimivuudesta testiympäristössä, mutta pitkällä aikavälillä testausdatan suuremmalla hyödyntämisellä pyritään myös tuotekehitys- ja testausajan lyhentämiseen sekä uusien oivallusten tekemiseen data-analyysin avulla. Kaikkien näiden toimenpiteiden yhteisenä tavoitteena on tukiasemaohjelmistojen laadun parantaminen.

Tukiasemien testausympäristön tuottama tieto voi olla rivimuotoista lokitietoa, kuten syslokit, CSV-tiedostot tai rakenteellista tietoa, kuten XML- ja JSON-tietomuodot. Opinnäytetyön alussa perehdytään näiden lokitiedostojen tietomuotoihin, koska niiden tunteminen on välttämätöntä tiedon analysoinnissa. Myös hakukoneisiin liittyviä yleisiä ominaisuuksia käsitellään työn alussa.

Opinnäytetyössä ei käsitellä tukiasemien testausprosessia, eikä syitä eri muotoiselle lokitiedolle vaan lähtökohtana työlle on, että testausprosessi tuottaa eri muotoista lokitietoa, jonka sisältämää tietoa on voitava analysoida helpommin. Työssä ei myöskään käsitellä hakukoneiden asennusprosessia palvelimille, koska ohjelmat olivat jo asennettuna palvelimille työtä aloitettaessa.

Opinnäytetyössä testataan kahta avoimenlähdekoodin hakukonetta: Elasticsearch ja Apache Solr. Työssä selvitetään eri tapoja miten tietoa voidaan viedä näiden hakukoneiden indekseihin, ja miten indeksoitua tietoa voidaan tämän jälkeen hyöndyntää.

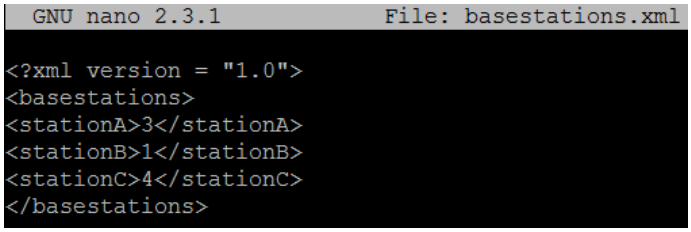
2 TESTAUSLOKIEN TIETOMUODOT

Lokitiedostot voivat olla hankalasti ymmärrettäviä ja eri tyyppisten lokien määrä voi hankaloittaa tilannetta entisestään. Toisaalta ne ovat hyödyllisiä ongelmia selvitettäessä. Lokitiedostojen tarkoituksena on auttaa selvittämään vikojen syitä ja korjaamaan niitä. Tämän lisäksi lokitiedostoista kerättävällä tiedolla voidaan saada tietoa muun muassa järjestelmän vakaudesta ja luokitella ongelmien syitä. (Splunk 2018, viitattu 7.2.2018.) Kappaleessa käsiteltäviä lokitiedostojen tietomuotoja ovat: XML, JSON, CSV ja SYSLOG.

2.1 XML

XML (Extensible Markup Language) on SGML-kieleen (Standard Generalized Markup Language) pohjautuva tiedon merkintäkieli. Merkintä tarkoittaa dokumenttiin lisättyä tietoa, jolla dokumentin sisältämälle tiedolle voidaan antaa merkitys. Merkinnällä tarkoitetaan elementtejä, joiden nimet ovat < > -merkkien sisällä kuten esimerkiksi <*yhteystiedot*>. Jokainen XML -elementti pitää sulkea päättävällä elementin nimellä, jolloin päättävän elementin nimen edessä on /-merkki, esimerkiksi <*sarjanumero*>*JK4567GH*</*sarjanumero*> tai tyhjän elementin ollessa kyseessä <*sarjanumero arvo=" JK4567GH" /*>. XML-kieli muistuttaa hieman HTML-merkintää, mutta selkeä ero on, että XML-kieltä käytetään tiedon organisointiin, kun taas HTML-merkinnällä määritetään dokumentin ulkoasu. Toinen ero on vielä, että XML-kielessä elementtien nimiä ei ole rajoitettu vaan elementit voidaan vapaasti nimetä kuvaamaan esimerkiksi elementin sisältämää tietoa. (Tutorialspoint 2018a, viitattu 8.2.2018.)

XML-elementti voi sisältää toisia XML-elementtejä, joita kutsutaan elementin lapsiksi. XML-dokumentti voi sisältää ainoastaan yhden juuri elementin (root element). XML-elementtien nimissä myös isot ja pienet kirjaimet merkitsevät eli elementti <*contact-info*> on eri kuin <*Contact-Info*>. (Tutorialspoint 2018b, viitattu 8.2.2018.) Kuviossa 1 on esimerkki XML-dokumentista.



KUVIO 1. XML-dokumentti

XML-dokumentin ensimmäisellä rivillä määritetään käytettävä XML-versio. Juuri elementti on <*basestations*>, jonka alla olevat elementit: <*stationA*>, <*stationB*> ja <*stationC*> ovat lapsielementtejä. Elementtien nimet ovat dokumentin merkintöjä ja numerot: 3,1 ja 4 ovat dokumentin sisältämää tietoa.

XML-elementille voidaan merkitä myös attribuutti, joka määrittelee elementin ominaisuuden. Esimerkiksi osoite -elementille voidaan määritellä koti ja työ ominaisuudet seuraavasti: <*osoite kategoria="koti"*>*…*</*osoite*> ja <*osoite kategoria="työ"*>*…*</*osoite*>. Attribuutteja käytetään, kun halutaan erotella samannimisiä elementtejä, eikä haluta luoda uusia elementtejä.(sama.)

XML-skeema

XML-skeemaa (XSD, XML Schema Definition) käytetään kuvaamaan ja määrittelemään XML-dokumentin sisältämät elementit ja niiden käyttämät tietotyypit. XML-skeemalla voidaan määrittää mitä tietotyyppiä XML-dokumentin kenttiin voidaan syöttää. Esimerkiksi elementtiin: <*xs:element name = "puhelin_nro" type = "xs:int" /*> voidaan tallentaa ainoastaan kokonaislukuja. Muita yleisiä elementin tietotyyppejä ovat: xs:boolean, xs:string ja xs:date. (Tutorialspoint 2018c, viitattu 8.2.2018.)

2.1.1 XML-dokumenttioliomalli

Dokumenttioliomalli (DOM, Document Object Model) esittää XML-dokumentin puurakenteena, joka sisältää solmukohtia (node). Puurakenteen ylimmäinen taso on juurisolmu (root node), jonka alla on elementtisolmuja (element node). Elementin sisältämä teksti on teksisolmu (text node) ja attribuutti on attribuuttisolmu (attribute node). XML-dokumentin puurakenteen solmuilla on hierarkinen suhde toisiinsa nähden. Näitä suhteita kutsutaan termeillä: isä (parent), lapsi (child) ja sisar (sibling). (w3schools 2018a, viitattu 9.2.2018.)

Kuvion 1 XML-dokumentissa elementti <*stationA*> on <*basestations*> elementin ensimmäinen lapsi elementti, <*stationB*> elementti on toinen lapsi ja <*stationC*> elementti on viimeinen lapsi ja <*basestations*> elementti on siis näiden isä -elementti ja samalla myös juurisolmu. Lapsisolmujen sisältö: ”3”, ”1” ja ”4” ovat tesktisolmuja. XML-dokumentin puurakenne määrittelee siis tavan miten XML-dokumentin solmuja voidaan ohjelmallisesti lukea, muuttaa, lisätä ja poistaa.

2.1.2 Xpath

Xpath (XML Path Language) on kyselykieli, jolla käydään XML-dokumentin puurakennetta läpi tietoja käsiteltäessä. Monet ohjelmointikielet voivat hyödyntää Xpath-kieltä XML-dokumentteja käsiteltäessä. Xpath käyttää polkuviittauksia valitsemaan XML-dokumentin sisältämiä solmuja. (w3schools 2018b, viitattu 9.2.2018.) Taulukossa 1 on Xpath-kielessä yleisesti käytettyjä polkulausekkeita.

TAULUKKO 1. Xpath-kielen polkulausekkeita (w3schools 2018c, viitattu 9.2.2018.)

Lauseke	Kuvaus
solmunimi (node)	Valitaan kaikki solmut annettulla solmun nimellä
/	Valinta alkaa ylimmältä tasolta eli juurisolmusta (Huom! Jos polku lauseke alkaa tällä merkillä niin kyseessä on aina absoluuttinen polku viittaus)
//	Valitaan nykyisen solmun sisältämät solmut valintaehdon mukaan
.	Valitaan nykyinen solmu
..	Valitaan nykyisen solmun isä solmu
@	Valitaan attribuutteja

Taulukossa 2 on muutamia esimerkkejä Xpath-polkulausekkeiden käytöstä kuvion 1 XML-dokumentin kanssa.

TAULUKKO 2. Xpath-polkulauseke esimerkkejä

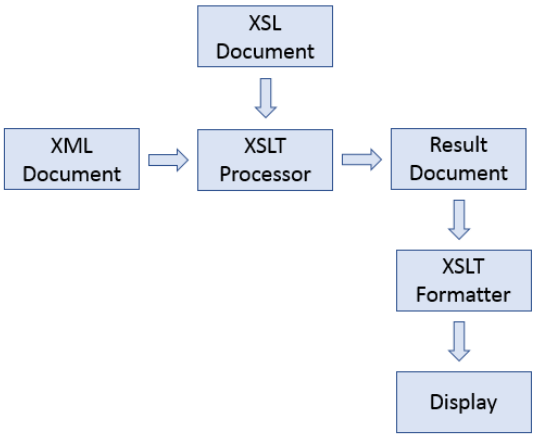
Lauseke	Tulos
basestations/stationA	Valitsee *stationA* nimisen elementin
//stationA	Valitsee *stationA* nimisen elementin
/basestations/*	Valitaan kaikki *basestations* elementin sisältämät lapsi elementit

2.1.3 Xslt

XSLT (Extensible Stylesheet Language) on merkintäkieli, jota voidaan käyttää XML-dokumentin automaattiseen muuntamiseen toiseen XML-dokumenttimuotoon tai esimerkiksi HTML-muotoon. XSLT-kieli käyttää XPATH-kieltä apuna muunnoksissa. (w3schools 2018d, viitattu 9.2.2018.)

Kuinka XSLT toimii

XSLT muunnos tapahtuu kirjoittamalla XSLT-tyylitiedosto, joka pitää sisällään muunnosohjeet kohteena olevaa XML-tiedostoa varten. XSLT-tyylitiedosto itsessään on XML muotoa ja tiedosto tallennetaan .XSL päätteisenä. XSLT-prosessori ottaa käyttöön XSLT-tiedoston ja muuntaa sen sisältämän tiedon perusteella kohteena olevan XML-tiedoston haluttuun muotoon (kuvio 2). (Tutorialspoint 2018d, viitattu 9.2.2018.)



KUVIO 2. XML-tiedoston muunnos XSLT-prosessorilla (sama)

XSLT-kielen etuna on, että se on riippumaton käytettävästä ohjelmointikielestä ja muunnosohjeet kirjoitetaan erilliseen XSL-tiedostoon, joka myös on XML dokumentti. Tulostettavan dokumentin ulkoasua voidaan helposti vaihdella tekemällä muutoksia XSL-tiedoston sisältämiin muunnosohjeisiin. (sama.)

2.2 Json

JSON (JavaScript Object Notation) on kevyt tiedon välittämiseen tarkoitettu ohjelmointikielistä riippumaton tietomuoto (json.org 2018, viitattu 9.2.2018).

JSON tietomuodon ominaisuuksia ovat:

- data tallennetaan nimi/arvo pareina
- data erotellaan pilkuilla
- aaltosuluilla merkitään objektit
- hakasuluilla merkitään taulukot.

(w3schools 2018e, viitattu 9.2.2018).

Kuviossa 3 on esimerkki JSON muotoisesta tiedosta, jossa on työntekijät niminen objekti ja joka sisältää taulukon, jossa on kolme työntekijää.

```
  GNU nano 2.3.1            File: employees.json

{"tyontekijat":[
        {"etunimi":"Jussi", "sukunimi":"Kolmonen"},
        {"etunimi":"Kari", "sukunimi":"Kaksonen"},
        {"etunimi":"Paula", "sukunimi":"Pihlaja"}
]}
```

KUVIO 3. Esimerkki JSON-tiedosto

2.3 Csv

CSV-tietomuoto (Comma Separated Values) on pilkuilla erotettua tietoa. CSV-tietomuodossa jokainen sarake on erotettu pilkulla ja jokainen uusi rivi merkitsee uutta tietuetta. Tiedosto, joka sisältää tämän muotoista tietoa, merkitään CSV-tiedostopäätteellä. CSV-tietomuotoa käytetään muun muassa tiedon siirtämiseen ohjelmien välillä. (Fisher 2018, viitattu 9.2.2018.) Kuviossa 4 on esimerkki CSV-tiedostosta, jossa on neljä saraketta ja otsikkorivin jälkeen ovat otsikon osoittamiin kenttiin sijoitetut arvot.

```
  GNU nano 2.3.1              File: osoitteet.csv

Etunimi,Sukunimi,Katu,PostiNro
Jussi,Yksi,Jalotie 3,54356
Jouni,Kolme,Kuusitie 1,34535
Paula,Pihlaja,Jalavatie 7,94530
```

KUVIO 4. Esimerkki CSV tietorakenteesta

2.4 Syslog

Syslog on erilaisten verkossa olevien laitteiden viestien lähettämiseen ja vastaanottamiseen kehitetty tietomuoto. Syslog-viestit kulkevat IP-verkoissa ja Syslog käyttää UDP-protokollaa (User Datagram Protocol) ja porttia 514 liikennöintiin. Syslog-viestit sisältävät muun muassa aikaleiman, tapahtumaviestin, ongelman vakavuuden tason (severity), IP-osoitteen ja ongelman kuvauksen. Syslog-viestit ovat yleisesti käytössä Linux-, Unix- ja MacOS-käyttöjärjestelmissä. Syslog-protokollan kirjoitti alun perin Eric Allman ja IETF (Internet Engineering Task Force) on määritellyt sen dokumentissa RFC 3164. Vuonna 2009 IETF standardoi Syslog protokollan dokumentissa RFC 5424. (Stackify 2017, viitattu 11.2.2018.)

Syslog-viestin muoto

Syslog-viestin muoto on määritelty dokumentissa RFC 5424. Syslog-viesti koostuu kolmesta osasta: HEADER, STRUCTURED-DATA (SD) ja MESSAGE. HEADER-osa sisältää kentät: PRIORITY, VERSION, TIMESTAMP, HOSTNAME, APPLICATION, PROCESS ID ja MESSAGE ID. Näiden jälkeen tulee STRUCTURED-DATA -kenttä hakasulkujen sisällä ja sisältää avain / arvo pareja. Viestin lopussa on MESSAGE-kenttä, joka sisältää viestin yksityiskohtaisemman kuvauksen. (sama.)

PRI -osa muodostuu kahdesta numero arvosta, joita ovat Facility ja Severity (Liite 1). Nämä arvot auttavat kategoroimaan viestin. Viestin PRI-osan arvo lasketaan Facility- ja Severity-arvoilla seuraavasti:

*(Facility arvo * 8) + Severity arvo = PRI*

Mitä alhaisempi PRI arvo on sitä korkeampi on prioriteetti. Tällä tavoin, kun kaavaan sijoitetaan esimerkiksi Kernel ongelmaa vastaava Facility numero 0 niin PRI arvoksi tulee pienempi arvo (korkea prioriteetti), kuin mitä saadaan Log alert Facility -lähteellä, jonka numero on 14, riippumatta Severity arvon suuruudesta. (Paessler 2018, viitattu 12.2.2018.)

Facility-arvo määrittää viestin lähteen ja sen arvo voi olla yksi 15:stä ennalta määritellystä arvosta 0-15 tai jokin paikallisesti määritellyistä arvoista väliltä 16-23. Severity-arvo määrittää viestin tärkeyden. (sama.)

Kuviossa 5 on esimerkki Syslog-viestistä

```
<34>1 2003-10-11T22:14:15.003Z mymachine.example.com su - ID47 - BOM'su root' fa
iled for lonvick on /dev/pts/8
```

KUVIO 5. Esimerkki Syslog viesti (Stackify 2017, viitattu 11.2.2018)

ja sitä vastaava tietomuoto (kuvio 6).

```
<priority>VERSION ISOTIMESTAMP HOSTNAME APPLICATION PID MESSAGEID STRUCTURED-DAT
A MSG
```

KUVIO 6. (sama)

2.4.1 Regex, säännöllinen lauseke

Regex-lausekkeet (Regular expressions, suom. säännölliset lausekkeet) liittyvät olennaisesti Syslog-viestien käsittelyyn, kun jäsennetään niiden sisältämää rivimuotoista tietoa.

Säännölliset lausekkeet ovat määriteltyjä tekstimalleja, joilla voidaan kuvata tietty merkkijono. Näillä malleilla voidaan tehdä esimerkiksi seuraavanlaisia asioita:

- osoitetiedoista voidaan tallentaa postinumero omaan kenttään
- tekstirivi voidaan jakaan pienempiin kenttiin jakamalla tekstirivi kenttiin, esimerkiksi käyttämällä jakamiseen rivillä esiintyviä pisteitä, pilkkuja tai välilyöntejä.

(Lopez & Romero 2014, 20–21.)

Säännölliset lausekkeet ovat peräisin 1950-luvulta, kun matemaatikko Stephen Kleene muotoili säännöllisten lausekkeiden määritelmän teoksessaan ”*Representation of events in nerve nets and finite automata*”. Tänä päivänä monissa ohjelmointikielissä on tuki säännöllisille lausekkeille. (sama.)

2.4.2 Säännöllisen lausekkeen muoto

Säännöllinen lauseke on tekstimalli, joka koostuu kirjaimista a – z, numeroista 0 – 9 ja kahdestatoista erikoismerkeistä (metacharacters), joita ovat pilkulla eroteltuina seuraavat merkit:

*\, ^, $, .,| , ?, *, + (,), [, {*

Jos näitä erikoismerkkejä esiintyy mallinnettavassa tekstissä, niin näiden merkkien eteen on laitettava tekstimallissa kenoviiva \-merkki, jolloin kenoviivan jälkeinen merkki tulkitaan tavallisena merkkinä eikä säännöllisenä lausekkeena. (Lopez & Romero 2014, 23,26.) Liitteessä 2 on lueteltu säännöllisessä lausekkeessa käytettäviä erikoismerkkejä ja esimerkkejä niiden käytöstä. Säännöllisiä lausekkeita voidaan tehdä myös lyhenteitä apuna käyttäen, jolloin käytetään takakeno \ -merkintää lyhenteen edessä (Liite 3).

Jos mallinnettavassa tekstissä esiintyy Regex-lausekkeissa käytettäviä erikoismerkkejä niin niiden eteen on laitettava kenoviiva \-merkki. Taulukossa 3 on esimerkkejä erikoismerkkien käytöstä Regex-lausekkeissa.

TAULUKKO 3. Esimerkkejä erikoismerkkien käytöstä Regex-lausekkeissa (ComputerHope 2018b, viitattu 13.2.2018)

Erikoismerkki	Regex lauseke	Merkkijonon vastaavuus
\\	\\	\
\^	\^{5}	^^^^^
\$	\$5	$5
\.	Yes\.	Yes.
*	typo*	typo*
\[	[3\[]	3,[
\]	\]	]

Regex lyhenteitä ja erikoismerkkejä käyttämällä voidaan tehdä monimutkaisia lausekkeita, joita voidaan käyttää monipuolisesti hyödyksi esimerkiksi lokitietoja käsiteltäessä. Seuraavaksi on esimerkki hieman monimutkaisemmasta Regex-lausekkeesta, jossa määritetään sähköpostiosoitteen muoto:

[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}

Tällä Regex-lausekkeella voidaan poimia tekstiriviltä sähköpostiosoite. Lauseke löytää siis esimerkiksi osoitteen test-one@mail.com. Regex-lauseke selitettynä:

[\w._%+-]+	Haettava merkkijono voi sisältää mitä tahansa merkkejä, jotka on merkitty hakasulkujen sisään. Lyhenne \w tarkoittaa merkkejä [a-zA-Z0-9_] ja näiden lisäksi merkkijono voi sisältää myös merkkejä: ._%+- Hakasulkujen ulkopuolella oleva + -merkki tarkoittaa, että hakasulkujen sisällä olevat merkit voivat esiintyä yhden tai useamman kerran.
@	Haettavassa merkkijonossa pitää olla @-merkki.
[\w.-]+	Merkkijono voi jälleen sisältää merkkejä [a-zA-Z0-9_] ja myös merkit . ja –. Merkki + hakasulkujen ulkopuolella tarkoittaa ettei merkkien määrää ole rajoitettu.
\.	Merkkijonossa pitää olla piste. Koska kyseessä on erikoismerkki, niin pisteen eteen on laitettava kenoviiva.
[a-zA-Z]{2,4}	Pisteen jälkeen merkkijonossa voi olla joko isoja tai pieniä kirjaimia ja niitä pitää olla vähintään 2 kpl, mutta enintään 4 kpl.(ComputerHope 2018a, viitattu 13.2.2018.)

Regex-lausekkeita muodostettaessa ja testattaessa kannattaa hyödyntää internetistä löytyviä työkaluja, jotka helpottavat huomattavasti lausekkeiden määrittelyä. Niissä voidaan kirjoittaa Regex-lauseke ja haluttu merkkijono, jota lausekkeen pitäisi vastata, ja työkalu ilmoittaa saman tien löydetyt vastaavuudet. Tämän jälkeen Regex-lauseketta voi testata miten se toimii omassa koodissa. Parhaimmat tällaiset työkalut myös selittävät käytetyn Regex-lausekkeen toiminnan.

3 HAKUKONEET

Lokitiedostojen sisältämä tieto tallennetaan hakukoneisiin dokumentteina. Kun tietoa haetaan hakukoneella, niin käytännössä hakukone käy läpi sen sisältämää dokumenttiluetteloa. Hakukoneen sisältämät dokumentit voivat muodostua palvelimelle tallennettuista tekstitiedostoista tai yleisemmin dokumentit voivat vastata sisältöä, kuten:

- myyntivaraston tuotteet
- kirjaston sisältämät kirjat
- yhteystiedot.

Dokumentti sisältää tietokenttiä (fields), jotka ovat nimettyjä dokumentin sisältämiä ominaisuuksia (attributes) ja koostuvat kahdesta osasta: kentän nimi ja kentän arvo. Voidaan sanoa, että hakukoneen dokumentti vastaa SQL-tietokannan riviä. Kun SQL-tietokannan taulu sisältää sarakkeita, joihin on tallennettu arvoja, niin dokumentti sisältää tietokenttiä, joissa on arvoja ja joiden tietotyyppi voi olla: teksti, kokonaisluku, liukuluku ja Boolean. (Turnbull & Berryman 2016, luku 2.1.1.)

3.1 Sisällön vienti hakukoneeseen

Tieto täytyy viedä ensin hakukoneeseen ennen kuin sille voidaan tehdä hakuja. Tieto viedään (extract) hakukoneeseen siitä sijainnista minne se on tallennettu. Tieto voi olla tallennettu esimerkiksi tietokantaan, tekstitiedostoihin tai www-sivuille. Kun tietoa ladataan (load) hakukoneeseen, niin se muunnetaan (transform) dokumenteiksi ja kentiksi analysointivaiheessa, jossa dokumenttien kenttien sisältämät arvot muunnetaan merkittyihin osiin (tokens). Tähän prosessiin viitataan usein lyhtenteellä ETL, joka tulee sanoista: extracting, transforming ja loading. (Turnbull & Berryman 2016, luku 2.1.4.)

Tieto tallennetaan hakukoneisiin dokumentteina, ja datan lähteet voivat olla monenlaisia. Käytännössä miten tieto viedään hakukoneeseen voi vaihdella suuresti riippuen datan lähteistä, joita voivat olla esimerkiksi tietokannat, www-sivut, tiedostojärjestelmät, MS Word-, PDF -, JSON -tiedostot jne. Tiedon viemiseen hakukoneeseen on monia menetelmiä, jotka vaihtelevat hakukoneittain.

Riippumatta vietävän tiedon muodosta, vientivaiheen lopputuloksena on dokumentteja, jotka lähetetään hakukoneeseen. (Turnbull & Berryman 2016, luku 2.3.1.)

3.2 Indeksointi

Analysointivaiheen jälkeen tiedon viemisen viimeisenä vaiheena on indeksointi, jossa vietävä tieto sijoitellaan hakukoneen tietorakenteisiin (inverted index). Kaikkea tietoa ei indeksoida vaan indeksoinnissa on päätettävä mikä tieto indeksoidaan ja mikä on käytettävä tietorakenne eli käytännössä tehdään päätös dokumentin indeksoitavista kentistä. Tällä on merkitystä, koska kenttä voi toimia hakukenttänä ainoastaan, jos se on indeksoitu. (Turnbull & Berryman 2016, luku 2.1.4.)

3.3 Hakukoneiden keskeiset tietorakenteet

Hakukone muodostuu muutamasta optimoidusta tietorakenteesta, jotka mahdollistavat nopean tiedonhaun. Hakukoneen ytimessä on tietorakenne nimeltä *inverted index*, joka on vastaavanlainen kuin kirjan sisällysluettelo. Inverted index muodostuu kahdesta pääosasta: sanaluettelosta (*term dictionary*) ja sijaintilistasta (*posting list*). Sanaluettelo on järjestetty lista kaikista sanoista (*term*), jotka esiintyvät tietyssä kentässä kaikissa dokumenteissä. Sanaluettelon jokaisen sanan sijaintitieto on tallennettu sijaintilistaan, josta selviää missä dokumentissa kukin sana esiintyy. (Turnbull & Berryman 2016, luku 2.2.1.) Seuraava esimerkki selvittää tätä asiaa lisää (ks. taulukko 4).

TAULUKKO 4. (sama)

Dokumentit	Sanaluettelo	Sijaintilista
0. Yksi auto, kaksi autoa, punainen auto, sininen auto	auto – 0	0 – 0,1,2
1. Sininen auto on paras auto	kaksi – 1	1 – 0
2. Punainen auto on toiseksi paras	on – 2	2 – 1,2
	paras – 3	3 – 1,2
	punainen – 4	4 – 0,2
	sininen – 5	5 – 0,1
	toiseksi - 6	6 – 2
	yksi – 7	7 - 0

Taulukosta 4 nähdään, että jos haetaan esimerkiksi dokumentteja, jotka sisältävät sanan ”paras”, niin ensin sanaluettelosta haetaan sanaa vastaava tunniste numero, joka sanalla ”paras” on numero kolme. Tämän jälkeen etsitään sijaintilistalta tunniste numeroa kolme vastaavat sijainnit, jotka ovat 1 ja 2, ja nähdään, että sana ”paras” esiintyy dokumenteissa 1 ja 2, mutta ei dokumentissa 0. (sama)

3.4 Lucene

Apache Lucene on tehokas, monipuolinen Java-kielellä kirjoitettu teksti hakukone kirjasto. Apache Lucene on avoimen lähdekoodin projekti, joka on julkaistu Apache lisenssillä ja on siten käytettävissä sekä kaupallisissa että avoimen lähdekoodin ohjelmissa. (Apache Lucene 2016, viitattu 16.2.2018.)

Apache Lucene hakukirjaston perustoimintoja ovat:

- dokumentin luonti hakukoneeseen vietävästä tiedosta
- dokumentin analysointi, jossa valitaan indeksoitava tieto
- dokumentin indeksointi, joka mahdollistaa tiedon haun haluttujen kenttien mukaan
- käyttöliittymä hakutoiminnolle, missä voidaan syöttää haettavaa tietoa
- kyselyn rakentaminen, haettavasta tiedosta tehdään kysely objekti
- haku kysely, käyttää kysely objektia, jolla indeksistä haetaan haettavan tiedon sisältämät dokumentit.

Apache Lucene -hakukirjasto tarjoaa siis edellä mainittuja ominaisuuksia hakukoneissa, jotka käyttävät tätä kirjastoa. Tiedon viemisen hakukoneeseen ja hakutulosten näyttämisen kukin hakukone hoitaa omalla tavallaan. (Tutorialspoint 2018e, viitattu 16.2.2018.)

3.5 Lucene hakukirjaston kyselytyypit

Lucene hakukirjastossa on eri tyyppisiä kysely vaihtoehtoja tiedon hakemiseen. Kyselyjä yhdistämällä on mahdollista tehdä monimutkaisia hakuja indeksin sisältämiin dokumentteihin.

TermQuery

Erilaisista kyselytyypeistä TermQuery on eniten käytetty. Tämän kyselyn tuloksena näytetään kaikki dokumentit, joissa esiintyy haettu sana tietyssä kentässä. (Apache Lucene 2018, viitattu 16.2.2018.)

BooleanQuery

Useita TermQuery-kyselyjä voidaan yhdistää Boolean-kyselyssä. Lucene hakukirjaston Boolean -kyselyssä käytetään seuraavia operaattoreita: SHOULD, MUST ja MUST NOT. Operaattoreita AND, OR ja NOT ei siis käytetä. Kun käytetään SHOULD-operaattoria, niin lausekkeen hakuehto voi esiintyä dokumentissa, mutta sitä ei vaadita. Dokumentit, jotka sisältävät SHOULD-hakuehdon, ovat kuitenkin korkeammalla sijalla tuloslistassa. MUST-operaattoria käytetään, kun vaaditaan, että dokumenteissa on oltava hakuehdossa määritelty nimike. MUST NOT -operaattorilla voidaan määrittää mitä haettavissa dokumenteissa ei saa esiintyä. (sama.)

PhraseQuery

Tällä kyselyllä haetaan dokumentteja, joissa esiintyy sanat annetussa järjestyksessä (sama).

TermRangeQuery

Hakuehdosssa voidaan määrittää raja-arvot minkälaisia dokumentteja kyselyllä halutaan löytää. Voidaan esimerkiksi määrittää, että halutaan löytää kaikki dokumentit, joissa tietyn kentän sanojen ensimmäinen kirjain on kirjaimien a ja c välillä. (sama.)

PointRangeQuery

Kyselyllä voidaan etsiä dokumentteja, joissa esiintyy tietyn suuruisia lukuarvoja. Tämä edellyttää, että numeroarvoja sisältävät kentät ovat indeksissä tietotyypiltään numeroita. (sama.)

PrefixQuery

Käytetään, kun halutaan löytää dokumentteja, joissa kenttä alkaa hakuehdossa määritellyllä sanalla (sama).

WildcardQuery

Tässä kyselyssä voidaan käyttää merkkejä * tai ?. Merkki * merkitsee nolla tai monta osumaa ja merkki ? merkitsee tarkalleen yhtä osumaa (sama).

RegexpQuery

Kyselyssä määritetään Regex-malli, jonka mukaisia dokumentteja halutaan löytää (sama).

FuzzyQuery

Kyselyssä haetaan dokumentteja, jotka sisältävät vastaavanlaisen sanan kuin hakuehdossa on määritelty. Sanojen samanlaisuuden arvioiminen perustuu Levenshtein nimiseen algoritmiin. Tämä haku löytää siis dokumentit, jos hakuehdossa olisi väärin kirjoitettu sana ”netwoking”, vaikka tätä sanaa ei ole indeksoitu. (sama.)

4 HADOOP

Radioverkkojärjestelmien testausympäristö tuottaa suuren määrän lokitietoa, mitä ei voida tallentaa käyttämällä perinteisiä ratkaisuja, kuten tiedon tallennus Linux- tai Windows-tiedostojärjestelmiin tai relaatiotietokantoihin. Suuren tietomäärän lisäksi lokitietoa voi kertyä nopeassa tahdissa ja sen rakenne voi vaihdella, jolloin lokitieto täyttää yleiset Big Data -määritykset.

IBM määrittelee Big Datan kolmella v-kirjaimella. Big Datalle on ominaista, että se on tietoa, jota kertyy nopeassa tahdissa (velocity), suuria määriä (volume) ja joka on monimuotoista (variety). Tällaista tietoa tulee muun muassa sensoreilta, erilaisilta laitteilta, lokitiedostoista ja sosiaalisesta mediasta usein reaaliajassa ja suuressa mittakaavassa. (IBM 2018, viitattu 27.3.2018.)

Hadoop on avoimen lähdekoodin Apache yhteisön Java-ohjelmointikielellä kirjoitettu sovellus, joka on vikasietoinen, skaalautuva ratkaisu ja tarkoitettu suurten tietomäärien hajautettuun käsittelyyn ja tallentamiseen. Hadoopissa suuri tietomassa voidaan tallentaa muutaman palvelimen ratkaisusta aina tuhansien koneiden klustereihin. (Tutorialspoint 2018f, viitattu 27.3.2018.)

4.1 HDFS-tiedostojärjestelmä

Hadoopin tiedostojärjestelmä on HDFS (Hadoop Distributed File System), mikä tarkoittaa vikasietoista hajautettua tiedostojärjestelmää. Tieto tallennetaan HDFS:ssä usealle palvelimelle (Data Nodes), ja jo tiedostojärjestelmän suunnittelussa on otettu huomioon, että palvelimia rikkoutuu. Järjestelmän vikasietoisuus toteutetaan Hadoop ohjelmassa, joka replikoi tietoa usealle palvelimelle. Palvelimen vikaantuessa Hadoop osaa hakea halutun tiedon toisilta palvelimilta. Tiedostojärjestelmissä tiedostot tallennetaan lohkoina (block). HDFS:ssä lohkon koko oletuksena on 64MB, mutta sitä voidaan kasvattaa tarpeen mukaan. (Tutorialspoint 2018g, viitattu 27.3.2018.) Taulukossa 5 on yleisiä HDFS-tiedostojärjestelmän kanssa käytettäviä Hadoop komentoja. Komennolla *hadoop fs -help* löytyy lisää tietoa Hadoop komennoista.

TAULUKKO 5. HDFS-tiedostojärjestelmän komentoja (Tutorialspoint 2018h, viitattu 27.3.2018)

Hadoop-komennon kuvaus	Hadoop-komento
HDFS:n tiedostoluettelon listaus	*hadoop fs -ls*
Hakemiston luonti HDFS:ään	*hadoop fs -mkdir /user/input*
Tiedoston vienti Linuxista HDFS:ään	*hadoop fs -put /home/file.txt /user/input*
Input hakemiston tiedostoluettelo	*hadoop fs -ls /user/input*
Tiedoston sisällön katsominen	*hadoop fs -cat /user/input/file.txt*
Tiedoston siirto HDFS:stä Linux-palvelimelle	*hadoop fs -get /user/input /home/hadoop_tp/*

4.2 WebHDFS

HDFS-tiedostojärjestelmää voidaan käyttää myös Hortonworks yrityksen kehittämän HDFS HTTP REST - rajapinnan (Representational State Transfer) kautta, jota kutsutaan nimellä WebHDFS. WebHDFS:n kanssa voidaan käyttää HTTP-operaatioita kuten GET, PUT, POST ja DELETE. Tämä mahdollistaa HDFS:n käytön useilla ohjelmointikielillä sekä yleisillä työkaluilla kuten curl ja wget. (Hortonworks 2017, viitattu 3.4.2018.) Linuxin *curl*-komennolla voidaan siirtää tietoa palvelimien välillä käyttämällä yhteyskäytäntöjä kuten HTTP, HTTPS ja FTP (Tutorialspoint 2018i, viitattu 3.4.2018). Edellisessä kappaleessa esitetyt Hadoop HDFS -komennot voidaan kirjoittaa WebHDFS:ää hyödyntäen curl-työkalulla seuraavasti:

HDFS:ssä olevan demo-kansion tiedostojen selailu:
curl -i 'http://localhost:50070/webhdfs/v1/demo/?user.name=hadoop&op=LISTSTATUS'

test nimisen hakemiston luonti demo-kansion alle:
curl -i -X PUT 'localhost:50070/webhdfs/v1/demo/test?user.name=hadoop&op=MKDIRS'

file.txt -tiedoston vienti Linuxista Hadoopiin test-kansion alle:
curl -i -L -T file.txt -X PUT 'http://localhost:50070/webhdfs/v1/demo/test/file.txt?user.name=hadoop&op=CREATE'

Hadoopissa test-kansiossa olevan file.txt -tiedoston sisällön katsominen:
curl -i -L 'http://localhost:50070/webhdfs/v1/demo/test/file.txt?user.name=hadoop&op=OPEN'
(Apache Hadoop 2017, viitattu 3.4.2018.)

5 ELASTICSEARCH

Elasticsearch on avoimen lähdekoodin Java-kielellä kirjoitettu teksti hakukone, joka pohjautuu Apache Lucene hakukone kirjaston käyttöön. Shay Banon aloitti Elasticsearch hakukoneen kehittämisen vuonna 2010. Tätä ennen hän oli tehnyt Compass nimisen hakukoneen, joka myös käytti Lucene kirjastoa. Lucene kirjaston päivittyminen tuolloin versioon 2.9 toi isoja muutoksia hakukoneen laajennettavuuteen liittyiviin asioihin, joten Shay Banon päätti kirjoittaa hakukoneen kokonaan uudelleen ja julkaisi sen siis Elasticsearch nimellä. (Gupta & Gupta 2017, luku 1.)

Elasticsearch hakukoneella voidaan tallentaa, hakea ja analysoida suuria tietomääriä nopeasti ja lähes reaaliajassa. Lokitietoja kerätään, jotta niitä voidaan analysoida ja työstää, jolloin niistä voidaan löytää kehityssuuntia, tilastoja, yhteenvetoja tai poikkeavuuksia. Kun tieto on tallennettu Elasticsearch hakukoneeseen, sille voidaan tehdä erilaisia hakuja, joilla voidaan saada aikaan edellä mainitun kaltaisia yhteenvetoja indeksoidusta tiedosta. (Elastic 2018a, viitattu 19.2.2018)

5.1 Elasticsearch hakukoneen peruskäsitteitä

Kappaleessa 3 kerrottiin hakukoneiden toimintaperiaatteesta yleisesti. Peruskäsitteiden tietäminen on oleellista hakukoneen toiminnan ymmärtämisen kannalta. Näitä käsitellään vielä seuraavaksi lyhyesti.

Indeksi

Indeksi on kokoelma samankaltaisia dokumentteja. Voi olla esimerkiksi asiakastietojen indeksi, tuoteluettelo indeksi, asiakastietojen indeksi jne. Indeksi nimetään (käytettävä pieniä kirjaimia) ja indeksin nimeä käytetään, kun kohdistetaan toimenpiteitä indeksin sisältämiin dokumentteihin. (Elastic 2018a, viitattu 19.2.2018.)

Tyyppi

Indeksi voi jakaantua yhteen tai useampaan tyyppiin (type). Tyyppi on looginen indeksin osa, jonka merkityksen voi päättää tarpeen mukaan. Yleensä saman tyypin dokumenteilla on yhteisiä kenttiä.

Esimerkiksi blogin sisältämä tieto voidaan tallentaa yhteen indeksiin, mutta indeksi voi jakautua eri tyyppeihin seuraavasti: käyttäjätiedot, blogi kirjoitukset ja kommentit. (sama.)

Dokumentti

Dokumentti on indeksoitavan tiedon perusyksikkö, esimerkiksi dokumentti, joka sisältää yksittäisen asiakkaan tiedot (sama).

Klusteri & node

Elasticsearch palvelimet (node), joita voi olla yksi tai useampi, muodostavat klusterin (cluster). Klusteri nimetään yksilöllisellä nimellä, mikä oletuksena on ”elasticsearch”. Jos Elasticsearch ilmentymiä (instance) on useita samassa verkossa, niin klusteri pitää aina nimetä uudella nimellä, koska muuten palvelimet voivat liittyä väärään klusteriin, koska palvelimia asennettaessa, jos klusterin nimeä ei anneta, palvelin liittyy automaattisesti elasticsearch nimiseen klusteriin. (Gupta & Gupta 2017, luku 2.)

Shard

Tallennettavia dokumentteja voi olla niin paljon, ettei yksittäinen palvelin (node) selviä tietomäärästä johtuen suorituskyky ongelmista, kuten vähäinen levytila, prosessointi teho jne. Tällaisessa tilanteessa tieto voidaan jakaa pienempiin osiin (shard). Jokainen shard on erillinen Apache Lucene indeksi ja näistä jokainen voi sijaita eri palvelimella. Hakuja tehtäessä hakukone lähettää kyselyn jokaiselle shard:lle ja yhdistää lopuksi hakutulokset. Tällainen tiedon jakaminen osiin voi parantaa myös suorituskykyä nopeuttamalla indeksointia. (Rogozinski & Kuc 2014, sivu 43.)

Replica

Vikasietoisuutta voidaan lisätä myös shard-kopioilla (replica). Hakukoneessa voi olla monia identtisiä shard-kopioita, joista yksi on automaattisesti pää shard ja toiset ovat shard kopioita. Jos pää shard menetetään jostain syystä, tällöin yhdestä shard kopiosta tulee uusi pää shard. (sama.)

5.2 Elasticsearch Stack

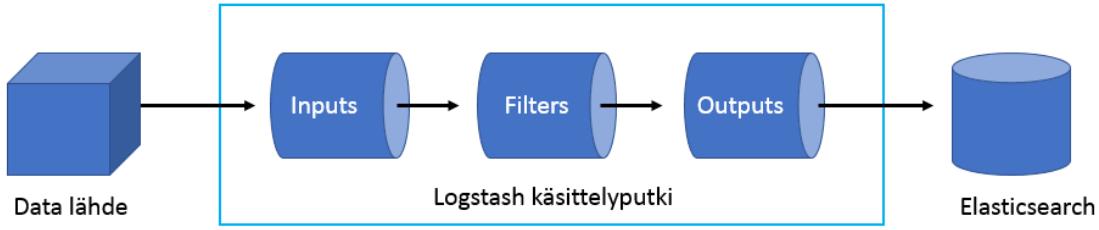
Kun Elasticsearch on saatu asennettua palvelimelle, seuraava toimenpide on tallentaa hakukoneen indeksiin tietoa, jotta sitä voidaan analysoida hakukoneen avulla.

Elastic -yrityksen kolme ohjelmaa: Elasticsearch, Logstash ja Kibana hoitavat asian yhdessä ja näitä kolmea ohjelmaa kutsutaan yleisesti ELK Stack -lyhenteellä. Tieto luetaan data lähteestä Logstashilla ja indeksoidaan Elasticsearch indeksiin, josta sitä voidaan hakea ja visualisoida Kibana ohjelmalla (kuvio 7). (Gupta & Gupta 2017, luku 1.)

KUVIO 7. ELK -Stack (sama)

5.3 Logstash

Logstash on avoimen lähdekoodin palvelimelle asennettava ohjelma, jolla voidaan samanaikaisesti viedä dataa useasta eri lähteestä Elasticsearch indeksiin. Data on usein tietomuodoltaan moninaista ja hajallaan eri järjestelmissä. Logstashissa on useita input-lisäosia (plugins) datan keskeytymättömään lukemiseen eri lähteistä. Datan lukemisen jälkeen Logstashin filter-lisäosat jäsentelevät (parse) tiedon muodostaen siitä tietokenttiä. Lopuksi Logstashin output-osiossa käsitelty tieto voidaan viedä esimerkiksi Elasticsearch indeksiin, mutta tämä ei ole ainoa vaihtoehto, sillä myös output-lisäosia on Logstashissa moneen tarkoitukseen. (Elastic 2018b, viitattu 21.2.2018) Kuviossa 8 nähdään miten tieto kulkee data lähteeltä Logstashin kautta Elasticsearchiin.



*KUVIO 8. Logstash käsittelyputki (*Elastic 2018c, viitattu *21.2.2018)*

Logstash asennuksen toimivuutta voidaan testata ajamalla Logstash asennuskansiossa komento:

bin/logstash -e 'input { stdin { } } output { stdout {} }'

Edellisessä komennossa e-lippu määrittelee, että Logstashin konfiguraatio syötetään komentorivillä heittomerkkien sisällä. Edellisessä esimerkissä Logstashin syöte tulee näppäimistöltä, stdin, joka tulostuu näytölle, stdout. Kun Logstash käynnistetään esimerkin komennolla niin Logstash käynnistyy ja odottaa syötettä näppäimistöltä, kun näytöllä lukee “Pipeline main started”. Kun kirjoitetaan hello, niin komentoriville tulostuu:

2018-02-21T11:19:28.732Z localhost hello

Logstash lisää aikaleiman ja IP-osoitteen tulosteeseen kirjoitetun tekstin eteen. Logstash ajo voidaan keskeyttää CTRL-D komennolla komentorivi näkymässä. (*Elastic 2018c, viitattu* 21.2.2018)

5.3.1 Logstash ohjelman konfiguraatiotiedosto

Edellä nähdyssä Logstash esimerkissä Logstash komento ajettiin e-parametrillä, joka merkitsi Logstashin käyttämää konfiguraatiota komentorivillä annettuna. Käytännössä kuitenkin tehdään yleensä aina erillinen konfiguraatiotiedosto, johon määritellään mitä lisäosia käytetään ja tarvittavat asetukset kullekin lisäosalle. Konfiguraatiotiedostoa käytettäessä e-parametrin sijasta käytetään tällöin komennossa f -parametriä. Tehdään kuviossa 9 näkyvä “*logstash-simple.conf*” niminen konfiguraatiotiedosto.

```
  GNU nano 2.3.1          File: logstash-simple.conf

input { stdin { } }
output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

KUVIO 9. Logstash konfiguraatiotiedosto (Elastic 2018d, viitattu 21.2.2018)

Konfiguraatiotiedosto voidaan tallentaa haluttuun paikkaan. Seuraava komento olettaa, että tiedosto on samassa paikassa kuin Logstashin käynnistyskomento ja komento ajetaan f-parametrillä:

bin/logstash -f logstash-simple.conf

Logstash käyttää nyt *"logstash-simple.conf"* nimistä konfiguraatiotiedostoa, johon on määritelty, että syöte tulee näppäimistöltä ja tieto viedään sekä Elasticsearch indeksiin että tulostetaan myös näytölle. (sama.)

5.3.2 Konfiguraatiotiedoston rakenne

Konfigurointitiedosto koostuu siis kolmesta osasta: input, filter ja output-osioista seuraavasti:

input {}

filter {}

output {}

Jokainen osio voi sisältää yhden tai useamman lisäosan (plugin). Jos osiossa käytetään useaa lisäosaa, niitä ajetaan siinä järjestyksessä kuin ne ovat tiedostossa. Jokaisella lisäosalla on myös omia erityisiä asetuksia, joita voi käyttää vain kyseisen lisäosan kanssa. Alla on esimerkki file-lisäosan käytöstä input osiossa:

```
input {
  file {
    path => "/var/log/messages"
    type => "syslog"
  }
}
```

Edellä nähdään, että file-lisäosa sisältää muutaman asetuksen: path ja type, jotka on merkitty aalto-suluilla { } merkittyyn lohkoon.

Tiedostossa voidaan käyttää erikoismerkkejä kuten # ja =>. Kommentti tehdään laittamalla tekstin eteen # -merkki ja kommentti voi sijaita myös rivin lopussa esim.:

```
input {      # tämä on kommentti
# tämä on toinen kommentti
```

Kun avaimeen tallennetaan arvo niin käytetään sijoitusoperaattoria => esim. key => value, numero => 44 tai name => “Hello world”. (Elastic 2018e, viitattu 21.2.2018.)

5.3.3 Logstash toimintaperiaate

Tieto käsitellään Logstashissa kolmessa vaiheessa: inputs -> filters -> outputs. Inputs-osiossa muodostuu tapahtumat (events), filters-osiossa käsitellään niitä ja outputs-osiossa ne siirretään eteenpäin. (Elastic 2018g, viitattu 22.2.2018.)

Esimerkiksi Syslog-tiedoston rivi voi olla tapahtuma. Kun rivi on luettu Logstashiin input-osiossa käyttämällä jotakin lisäosaa, niin siitä tulee tapahtuma, jota käsitellään filter-osiossa, jossa sisään luetusta tiedosta muodostetaan kenttiä. Tämä tarkoittaa, että kenttiin voidaan viitata vasta filter- ja output-osioissa, koska input-osiossa niitä ei ole vielä olemassa. (sama.)

Kenttäviittaukset

Kenttiin viitataan laittamalla kentän nimi hakasulkuihin *[fieldname].* Jos kyseessä on ylimmän tason kenttä, voidaan kirjoittaa *fieldname* ilman hakasulkuja. Jos kyseessä on hierarkinen, sisennetty kenttä, niin on käytettävä hakasulkuja kentän nimessä: *[top-level field][nested field].* Esimerkkinä hierarkisesta kentästä seuraava Json-muodossa oleva tieto:

```
{
   "response": {
   "status": 200,
   "bytes": 523534
  },
  "ua": {
   "os": "Windows 10"
  }
}
```

Jos halutaan viitata os-kenttään, niin viittaus on muotoa: *[ua][os].* Kenttien arvoihin voidaan viitata myös ns. sprintf-muodossa. Tämä muoto mahdollistaa kenttäviittaukset muiden merkkijonojen sisältä. Esimerkiksi *statsd output* -lisäosassa on *increment* asetus, joka pitää lukua apache lokeista status koodin mukaan:

```
output {
  statsd {
    increment => "apache.%{[response][status]}"
  }
}
```

(sama.)

Ehtolausekkeet

Logstashissa on käytössä ehtolausekkeet samaan tapaan kuin ohjelmointikielissä:

```
if LAUSEKE {
  ...
} else if LAUSEKE {
  ...
} else {
  ...
}
```

Lausekkeet voivat siltää vertailuja, boolean logiikkaa jne.

Seuraavat vertailuoperaattorit ovat käytössä:

- yhtäsuuruus: ==, !=, <, >, <=, >=
- regexp: =~, !~ (lausekkeessa oikealla puolella olevaa mallia testataan vasemmalla olevaa arvoa vasten)
- lisäys: in, not in.

Boolean operaattoreista käytössä ovat: and, or, nand, xor.

Lausekkeet voivat sisältää toisia lausekkeita ja lausekkeen toiminnon voi kääntää (negate) ! -operaattorilla ja lausekkeita voi ryhmitellä käyttämällä sulkeita (…..). (sama.)

5.3.4 Logstash lisäosat

Logstash palvelimelle asennetut lisäosat (plugins) voidaan tarkistaa komennolla (-verbose tarkennin näyttää myös lisäosien versiotiedot): *bin/logstash-plugin list -verbose.* Logstash-plugin -komennosta saa lisätietoa muun muassa kuinka lisäosia asennetaan komennolla: *bin/logstash-plugin -h.*

Logstashin lisäosat tallennetaan ja niitä ylläpidetään Ruby yhteisön -verkkosivulla (Ruby community 2018, viitattu 11.4.2018). Kun komennolla *bin/logstash-plugin install* asennetaan lisäosa, niin tarvitaan toimiva internet yhteys ja lisäosa asennetaan tuolta sivustolta. Esimerkiksi logstash-input-http -lisäosa asennetaan seuraavasti: *bin/logstash-plugin install logstash-input-http.* Lisäosa voidaan poistaa käytöstä komennolla: *bin/logstash-plugin uninstall logstash-input-http.*

Jos esimerkiksi logstash-input-http -lisäosa on jo asennettu, mutta sen versio on vanhentunut, se voidaan päivittää komennolla: *bin/logstash-plugin update logstash-input-http.* Jos halutaan päivittää kaikki lisäosat samalla kertaa, niin se tapahtuu komennolla: *bin/logstash-plugin update.* (Sharma 2016, luku 9.)

Input lisäosat

Taulukossa 6 on viisi yleisesti käytettyä Logstash konfiguraatiotiedosoton input-osion lisäosaa.

TAULUKKO 6. Logstash Input lisäosia (Elastic 2018h, viitattu 22.2.2018)

Lisäosa	Kuvaus	Lisäosan asennustiedoston nimi
beats {}	Vastaanottaa tapahtumia Elasticin Beats -ohjelmilta	logstash-input-beats
exec {}	Vastaanottaa komentotulkin tulosteen tapahtumana	logstash-input-exec
file {}	Muodostaa tapahtumia tiedostoa lukemalla	logstash-input-file
http_poller {}	Muodostaa tapahtumia HTTP API -rajapinnan kautta	logstash-input-http_poller
udp {}	Muodostaa tapahtumia verkon yli UDP-protokollalla	logstash-input-udp

Filter lisäosat

Taulukossa 7 on viisi yleisesti käytettyä Logstash konfiguraatiotiedoston filter-osion lisäosaa.

TAULUKKO 7. Logstash Filter lisäosia (Elastic 2018i, viitattu 22.2.2018)

Lisäosa	Kuvaus	Lisäosan asennustiedoston nimi
csv {}	Jäsentää pilkuilla erotellun tiedon kentiksi	logstash-filter-csv
date {}	Muuttaa merkkimuotoisen ajan tapahtuman aikaleimaksi	logstash-filter-date
dissect {}	Muuttaa erilaisilla välimerkeillä erotellun tiedon kentiksi	logstash-filter-dissect
grok {}	Jäsentää tietoa kentiksi regex-sääntöjä hyödyntäen	logstash-filter-grok
xml {}	Jäsentää XML-muotoisen tiedon kentiksi	logstash-filter-xml

Output lisäosat

Taulukossa 8 on viisi yleisesti käytettyä Logstash konfiguraatiotiedoston output-osion lisäosaa.

TAULUKKO 8. Logstash Output lisäosia (Elastic 2018j, viitattu 22.2.2018)

Lisäosa	Kuvaus	Lisäosan asennustiedoston nimi
elasticsearch {}	Tallentaa tapahtumia Elasticsearchin indeksiin	logstash-output-elasticsearch
file {}	Tallentaa tapahtumia tiedostoon	logstash-output-file
stdout {}	Tulostaa tapahtumia oletus ulostuloon	logstash-output-stdout
mongodb {}	Kirjoittaa tapahtumia MongoDB tietokantaan	logstash-output-mongodb
webhdfs	Tallentaa tapahtumia Hadoopiin	logstash-output-webhdfs

Koodekit

Logstashin input- ja output-osioissa on käytössä koodekkeja (codecs). Input-osion koodekit tulkitsevat sisään tulevan datan. Output-osion koodekeilla data voidaan koodata ennenkuin data lähtee output-osioista. Koodekkien käyttäminen voi vähentää tarvettaa käyttää lisäosia filter-osiossa. Taulukossa 9 on viisi yleisesti käytettyä koodekkia Logstash konfiguraatiotiedostossa.

TAULUKKO 9. Logstashin koodekki-lisäosia (Elastic 2018f, viitattu 21.2.2018)

Lisäosa	Kuvaus	Lisäosan asennustiedoston nimi
json {}	Käytetään Json-taulukkojen lukemiseen	logstash-codec-json
json_lines {}	Käytetään Json muotoisen striimin lukemiseen	logstash-codec-json_lines
line {}	Käytetään rivimuotoisen tekstitiedon lukemiseen	logstash-codec-line
multiline {}	Käytetään yhdistämään useita rivejä tapahtumaksi	logstash-codec-multiline
rubydebug {}	Käytetään output-osiossa tulostuksen muotoiluun	logstash-codec-rubydebug

Esimerkki koodekin käytöstä:

```
output {
    stdout { codec => rubydebug}
}
```

5.4 Logstash esimerkki, XML-tiedoston vieminen indeksiin

Edellisessä kappaleessa kerrottiin Logstashin konfiguraatiotiedostosta ja siinä käytettävistä lisäosista. Seuraavaksi esitetään käytännön esimerkki siitä, kuinka tietoa voidaan viedä Elasticsearch hakukoneen indeksiin Logstash ohjelmaa ja sen lisäosia käyttämällä.

Alla on esimerkki (kuvio 10) tukiaseman testaustiedostosta, joka on XML muotoa. Tiedostoa on esimerkissä lyhennetty paljon käytännön syistä. Esimerkissä on *Omes* niminen juuri elementti. *PMMOResult*-elementtejä voi tiedostossa olla vaihteleva määrä ja myös elementti *NE-LNBTS_1.0* voi sisältää vaihtelevan määrän lapsielementtejä eli käytännössä tiedosto voi sisältää kymmeniä *PMMOResult*-elementtejä ja kukin *NE-LNBTS_1.0*-elementti voi sisältää satoja lapsielementtejä (= laskuri arvoja: M8012C0...M8012C178). Tieto halutaan viedä Elasticsearch indeksiin seuraavalla tavalla (kuvio 10):

- jokainen *NE-LNBTS_1.0*-elementin laskuri arvo halutaan tallentaa Name / Value pariksi
- jokaiseen laskuri arvoon sidotaan myös *localMoid*-elementin arvo sekä *startTime*-attribuutin arvo.

```
<?xml version="1.0"?>
<OMeS xmlns="pm/cnf_lte_nsn.1.0.xsd">
<PMSetup startTime="2017-10-09T15:45:00.000+03:00:00" interval="15">
    <PMMOResult>
      <MO>
        <baseId>NE-MRBTS-19203</baseId>
        <localMoid>DN:NE-LNBTS-19203/LNCEL-11</localMoid>
      </MO>
      <MO>
        <DN>PLMN-PLMN/MCC-244/MNC-09</DN>
      </MO>
      <NE-LNBTS_1.0 measurementType="LTE_Cell_Throughput">
        <M8012C0>1</M8012C0>
        <M8012C1>11</M8012C1>
        <M8012C17>21</M8012C17>
        <M8012C18>3</M8012C18>
        <M8012C134>7</M8012C134>
        <M8012C178>15</M8012C178>
      </NE-LNBTS_1.0>
    </PMMOResult>
    <PMMOResult>
      <MO>
        <baseId>NE-MRBTS-19203</baseId>
        <localMoid>DN:NE-LNBTS-19203/LNCEL-12</localMoid>
      </MO>
      <MO>
        <DN>PLMN-PLMN/MCC-244/MNC-09</DN>
      </MO>
      <NE-LNBTS_1.0 measurementType="LTE_Cell_Throughput">
        <M8012C0>4</M8012C0>
        <M8012C1>17</M8012C1>
        <M8012C2>11</M8012C2>
        <M8012C178>12</M8012C178>
      </NE-LNBTS_1.0>
    </PMMOResult>
  </PMSetup>
</OMeS>
```

KUVIO 10. Kuvaruutukaappaus tiedostosta koe3a_omes.xml

Kuviossa 11 nähdään miltä tieto näyttää Elasticsearchin indeksissä, kun se on käsitelty Logstashilla. Kuvassa näkyy XML-tiedoston ensimmäinen laskuri arvo Logstashilla kenttiin jaoteltuna: <M8012C0>11</M8012C0> sekä muita Logstashin Filter-osiossa luotuja kenttiä, kuten *MO_type, Time, localMoid, Name, NE_type, Value, MO_id* ja *BTS_id*. Tieto on Elasticsearchin indeksissä JSON-muodossa ja alla näkyy yksi laskuri arvo, joka on JSON-objekti ja muut laskuri arvot muodostavat samanlaisia objekteja, jotka yhdessä muodostavat JSON-taulukon.


```
{
  "_index": "koe3a_omes_xml",
  "_type": "pm_omes",
  "_id": "AWHR2ZVDoMZ7-pQIwLPV",
  "_score": 1,
  "_source": {
    "MO_type": "LNCEL",
    "Time": "2017-10-09T15:45:00.000+03:00:00",
    "type": "pm_omes",
    "localMoid": "DN:NE-LNBTS-19203/LNCEL-11",
    "Name": "M8012C0",
    "path": "/home/hadoop/logstash-5.5.1/data/a/data/koe3a_omes.xml",
    "@timestamp": "2018-02-26T11:22:31.832Z",
    "NE_type": "LNBTS",
    "@version": "1",
    "host": "10.105.43.118",
    "Value": 11,
    "MO_id": 11,
    "BTS_id": 19203
  }
```


KUVIO 11. Elasticsearch indeksi

Seuraavaksi esitetään, kuinka kuvion 10 XML-tiedosto viedään Logstash ohjelmaa käyttämällä Elasticsearch indeksiin niin, että siitä muodostuu kuvion 11 kaltaisia JSON-objekteja.

Tehdään Logstash konfiguraatiotiedosto, johon tulee *input{}-, filter{}-* ja *output{}*-osiot. Tiedosto voidaan nimetä vapaasti ja tallentaa haluttuun paikkaan. Nimetään tiedosto *koe3a_omes_xml.conf* nimiseksi. Kuviossa 12 on konfiguraatiotiedoston input-osio.

```
input {

        file {
                path => "/home/hadoop/logstash-5.5.1/data/a/data/*.xml"
                sincedb_path => "/dev/null"
                start_position => "beginning"
                type => "pm_omes"
                        codec => multiline {
                                pattern => "<PMMOResult"
                                negate => "true" #negaten default arvo on false - kaytetaan PMMOResult elementin kanssa
                                what => "previous"

                        }
      } #end of file

} ## end of input
```

KUVIO 12. Logstash konfiguraatiotiedoston input-osio

Input-osiossa on käytetty *file{}*-lisäosaa tiedon lukemiseen. *File*-lisäosalla luetaan tiedostoa rivi kerrallaan. *Path*-asetuksella määritetään polku mistä tietoa luetaan. Asetuksella *sincedb_path* voidaan määrittää tietokanta, jossa pidetään kirjaa luetuista tiedostoista, niin ettei tietoa lueta useaan kertaan indeksiin. Esimerkissä tämä asetus ei ole käytössä, joten siitä syystä on käytetty hakemiston nimeä, jota ei ole olemassa eli */dev/null*. Asetuksella *start_position* kerrotaan, että tiedostoa aletaan lukemaan alusta. *Type*-asetuksella luodaan *type* niminen kenttä, jonka arvo on "*pm_omes*". Tätä kenttää voidaan hyödyntää esimerkiksi ehtolausekkeissa, jos samalla konfiguraatiotiedostolla käsitellään monenmuotoista dataa. *File*-lisäosassa käytetään vielä tässä tapauksessa *multiline* nimistä koodekkia. Logstashin konfiguraatiotiedostossa kommenttitiedon eteen laitetaan #-merkki.

File-lisäosa lukee tiedostoa rivi kerrallaan ja muodostaa aina luetusta rivistä tapahtuman. XML-tieto on rakenteellista tietoa, jossa yksittäinen rivi ei ole merkityksellinen vaan on luettava kokonaisia XML-elementtejä kerralla käsiteltäväksi. Tämä saadaan aikaan käyttämällä *multiline*-koodekkia. Multiline-koodekissa *pattern => "<PMMOResult"* -asetuksella määritetään XML-elementin alku, josta muodostetaan yksittäinen tapahtuma. Asetuksella *negate => "true"* saadaan aikaan, että *pattern => "<PMMOResult"* määrityksen jälkeiset rivit, jotka eivät vastaa tätä mallia kuuluvat samaan tapahtumaan ja asetus *what => "previous"* liittää nämä rivit yhteen aina edellisen rivin kanssa. Koodekin käyttö saa aikaan kuvion 13 mukaisen tapahtuman Logstashiin.

```
    "message": """
<PMMOResult>
  <MO>
    <baseId>NE-MRBTS-19203</baseId>
    <localMoid>DN:NE-LNBTS-19203/LNCEL-11</localMoid>
  </MO>
  <MO>
    <DN>PLMN-PLMN/MCC-244/MNC-09</DN>
  </MO>
  <NE-LNBTS_1.0 measurementType="LTE_Cell_Throughput">
    <M8012C0>11</M8012C0>
    <M8012C1>11</M8012C1>
    <M8012C17>11</M8012C17>
    <M8012C18>11</M8012C18>
    <M8012C134>11</M8012C134>
    <M8012C178>11</M8012C178>
  </NE-LNBTS_1.0>
</PMMOResult>
```

KUVIO 13. Logstash tapahtuma

XML-tiedoston sisältö otetaan käsittelyyn Logstashin filter-osiossa. Kuviossa 14 näkyy filter -osion alku, jossa on *if – else* lauseke. *If*-lauseessa testataan, että jos sisään luettu viesti sisältää merkki-

jonon ”*startTime*”, niin tämän jälkeen käytetään *xml {}* filtteriä, jonka tietolähteenä kyseinen viesti on. *Xpath*-komennolla haetaan *startTime*-attribuutin arvo *Aika* nimiseen muuttujaan. *Xpath*-komennolla mittausajankohta menee taulukkomuotoon ja *mutate – replace* kohdassa aika tallennetaan tästä taulukosta *Aika1* nimiseen väliaikaiseen tietokenttään. Tämä kenttä ei ole käytettävissä, kun käsitellään *PMMOResult*-elementtejä, koska *Aika1* on kenttä eikä muuttuja ja on siten käytettävissä ainoastaan tässä *if*-lausekkeessa, joka suoritetaan vain kerran tiedostoa käsiteltäessä (*startTime*-attribuutti esiintyy vain kerran XML-tiedoston alussa). Tästä syystä käytetään *Ruby*-filtteriä, jossa ajetaan Ruby koodia globaalin *“vara”* nimisen muuttujan tekoon, joka on käytettävissä Logstashin tapahtumien välillä.

If-lausekkeen *else*-haarassa tehdään *PMMOResult* niminen kenttä, joka sisältää samannimisen elementin sisällön kokonaisuudessaan XML-tiedoston elementtejä käsiteltäessä.

```
filter {
       if "startTime" in [message] {
               xml {
                       source => "message"
                       store_xml => "false" # true is default value
                       xpath => ["//@startTime","Aika"] # viedaan xml -tiedoston alussa olevan startTime -attribuutin arvo Aika -kenttaan
               }
               mutate {
                       replace => {
                               "Aika1" => "%{[Aika][0]}"
                      # muutetaan Aika -taulukko Aika1 kentta nimi muotoon joka on valiaikainen kentta
                       }
               }

               ruby {
                       init => "@@vara = ''" # tehdaan vara -niminen globaali muuttuja eventien valille
                       code => "@@vara = event.get('Aika1')" # viedaan Aika1-kentan arvo valiaikaiseen vara muuttujaan

               }
       } else {

               xml {
                       source => "message"
                       store_xml => "false"
                       xpath => ["/PMMOResult","PMMOResult"] # tehdaan PMMOResult kentta joka sis ko elementin sisallon
               }
       }
```

KUVIO 14. Logstash konfiguraatiotiedoston filter-osion alku

Filter-osio jatkuu kuviossa 15. *If*-lausekkeessa käytetään XML-suodatinta, jonka tietolähteenä on nyt *PMMOResult*-kenttä, joka luotiin edellä. Xpath-komennoilla haetaan tietoa XML-dokumentin DOM-rakenteesta, joka on *PMMOResult*-elementin sisältö. Xpath-komento *xpath => [“/PMMOResult/MO/localMoid/text()”,”localMoid”]* hakee *localMoid*-elementin sisällön kokonaisuudessaan *localMoid* nimiseen kenttään esim. *<localMoid>DN:NE-LNBTS-19203/LNCEL-11</localMoid>*. Seuraava xpath-komento: *xpath => [“/PMMOResult/NE-LNBTS_1.0/*”,”measurementType”]* hakee elementin *NE-LNBTS_1.0* kaikki lapsielementit *measurementType*-kenttään. Tämän jälkeen split-lisäosalla muodostetaan *measurementType*-kentän jokaisesta arvosta erillinen tapahtuma, jolloin kentän sisältö on esim. *<M8012C0>11</M8012C0>*. Seuraavaksi käytetään Dissect-lisäosaa tämän kentän jakamiseen

kahteen kenttään *"Name"* ja *"Value".* Dissect-lisäosassa käytetään välimerkkejä, joiden mukaan tieto jaetaan eri kenttiin. Mutate-lohkossa poistetaan väliaikaisia kenttiä, joita ei haluta tallentaa indeksiin, tehdään *localMoid* niminen kenttä ja muutetaan tiettyjen kenttien tietotyyppejä numeeriseen muotoon, jotka ilman tätä muunnosta olisivat merkkijonoja indeksissä.

```
if [PMMOResult] { # tarkistetaan onko kentta olemassa
        split {
                field => "PMMOResult" # xml tiedosto jaetaan edella muodostetun PMMOResult-kentan mukaan tapahtumiin
        }
        xml {
                source => "PMMOResult"  # PMMOResult -kentan sisaltama xml-koodi toimii taman xml-filtterin lahteena
               store_xml => "false"
                xpath => ["/PMMOResult/MO/localMoid/text()","localMoid"]
                # xpath -komentoa kaytetaan PMMOResult-kentan xml-koodille
                xpath => ["/PMMOResult/NE-LNBTS_1.0/*","measurementType"]
                # tallennetaan kaikki mittausarvot measurementType -kenttaan
        }
        split {
                field => "measurementType" # jaetaan kaikki mittausarvot omiin tapahtumiin
        }
        dissect {
                mapping => {
                        "measurementType" => "<%{Name}>%{Value}<%{}\n"
                        # jokainen mittausrvo rivi tallennetaan Name ja Value -kenttiin
                }
        }
        mutate {
                remove_field => ["message","tags","measurementType","PMMOResult"]
                # poistetaan valiaikaisia kenttia joita ei tarvita indeksissa

                replace => {
                        "localMoid" => "%{[localMoid][0]}"
                        # muutetaan localMoid -taulukko nimi kentta muotoon localMoid
                }
                convert => {
                        "Value" => "integer" # muutetaan Value-kentta numeeriseksi
                }
        } # end of mutate
```

KUVIO 15. Logstash konfiguraatiotiedoston filter-osion keskikohta

Kuviossa 16 on Filter-osion loppuosa. Dissect-lisäosalla kenttä *localMoid*, jonka sisältö on esim. *<localMoid>DN:NE-LNBTS-19203/LNCEL-11</localMoid>,* jaetaan neljään eri kenttään: *NE_type, BTS_id, MO_type* ja *MO_id*.

Ruby-lohkossa tallennetaan globaalin *vara*-muuttujan arvo Time-kenttään. Lopuksi tarkistetaan, että jos Filter-osion käsittelyssä on XML-tiedoston rivit, joissa on merkkijonot *"OMeS"* tai *"?xml version",* niin nämä rivit poistetaan Logstashin käsittelystä, eikä rivien sisältämää tietoa viedä indeksiin.

Jokainen mittausarvo viedään edellä kuvatulla tavalla erillisinä tapahtumina indeksiin, ja kun jokaiseen mittausarvoon liitetään vielä aika ja muita kenttiä, niin tiedolle voidaan tehdä hakuja tämän jälkeen.

```
            dissect {
                    mapping => {
                            "localMoid" => "%{}-%{NE_type}-%{BTS_id}/%{MO_type}-%{MO_id}"
                            # jaetaan localMoid -kentta neljaan eri kenttaan
                    }
             }
             mutate {
                     convert => {
                            "MO_id" => "integer"
                            "BTS_id" => "integer"
                    }
             }
             ruby {
                    code => "event.set('Time',@@vara)" # tallennetaan globaalimuuttujan vara arvo Time kenttaan
             }
      } # if loppu
      if "OMeS" in [message] or "?xml version" in [message] {  drop { } }
      # poistetaan xml tiedoston riveja Logstash filtterin kasittelysta
} ## end of filter
```

KUVIO 16. Logstash konfiguraatiotiedoston filter-osion loppuosa

Kuviossa 17 näkyy konfiguraatiotiedoston loppuosa. Output-osiossa on käytössä kaksi lisäosaa: stdout{}- ja elasticsearch{}-lisäosat. Stdout-lisäosa tulostaa näytölle käyttäen rubydebug-koodekkia mitä tulee ulos Logstashin käsittelyputkesta. Koodekki vaikuttaa missä muodossa tieto tulostuu näytölle. Vaihtoehtoisesti voidaan tulostaa myös JSON-muodossa, jolloin koodekki on: *codec => json*. Näytölle tulostaminen on hyödyllistä, kun testataan, että tuleeko Logstashin käsittelystä halutun muotoista tietoa. Kun Logstash on todettu toimivaksi, stdout-lisäosa voidaan ottaa pois käytöstä ja tallentaa tieto ainoastaan Elasticsearch hakukoneen indeksiin elasticsearch {} -output lisäosalla. Tämä lisäosa tallentaa tiedon indeksiin käyttäen http-protokollaa. Hosts-kohdassa näkyvän localhostin tilalla tulee käyttää Elasticsearch -palvelimen IP-osoitetta. Viimeisenä kohtana lisäosassa annetaan indeksin nimi, johon tieto tallennetaan.

```
output {

        stdout { codec => rubydebug  }

        elasticsearch {
                hosts => "http://localhost:9200"
                index=> "koe3a_omes_xml"

        }

} ## end of output
```

KUVIO 17. Logstash konfiguraatiotiedoston output-osio

Edellä tehtiin *koe3a_omes_xml.conf* niminen konfiguraatiotiedosto. Tiedosto otetaan käyttöön komennolla:

bin/logstash -f data/a/conf/koe3a_omes_xml.conf

Tällä komennolla luetaan kuviossa 12 näkyvän path määrityksen osoittamasta sijainnista tiedostoja Logstashiin käsiteltäväksi. XML-tiedoston jokainen mittausarvo viedään omana dokumenttina indeksiin ja Logstashin komentorivillä Rubydebug-koodekkia käytettäessä tämä näkyy myös näytölle tulostettuna (kuvio 18).

```
Logstash API endpoint {:port=>9600}
{
       "MO_type" => "LNCEL",
          "Time" => "2017-10-09T15:45:00.000+03:00:00",
          "type" => "pm_omes",
     "localMoid" => "DN:NE-LNBTS-19203/LNCEL-11",
          "Name" => "M8012C0",
          "path" => "/home/hadoop/logstash-5.5.1/data/a/data/koe3a_omes.xml",
    "@timestamp" => 2018-03-23T07:15:10.724Z,
       "NE_type" => "LNBTS",
      "@version" => "1",

         "Value" => 1,
         "MO_id" => 11,
        "BTS_id" => 19203
}
```

KUVIO 18. XML-dokumentin ensimmäinen mittausarvo Logstashin komentorivillä Rubydebug-koodekilla tulostettuna

5.5 Logstash esimerkki, CSV-tiedoston luku indeksiin Hadoop tiedostojärjestelmästä

Hadoopiin on tallennettu *test.csv* niminen tiedosto, joka halutaan viedä Logstash ohjelmalla Elasticsearch hakukoneen indeksiin (kuvio 19).

```
[hadoop@hadoopnode1 logstash-5.5.1]$hadoop fs -ls /demo/csv1
Found 1 items
-rw-r--r--   1 hadoop supergroup        740 2018-03-27 17:54 /demo/csv1/test.csv
[hadoop@hadoopnode1 logstash-5.5.1]$
```

KUVIO 19. CSV-tiedosto tallennettuna Hadoop:in HDFS-tiedostojärjestelmään

CSV-tiedoston sisältö näkyy kuviossa 20.

```
  GNU nano 2.3.1                          File: test.csv

COMMAND,RSS_AVG,RSS_AVG_MIN,RSS_AVG_MAX,AVG_CPU,CPU_AVG_MIN,CPU_AVG_MAX,HW_TYPE,TEST_TYPE,NR_OF_MEAS
HWR,22354.135618,14352,18704,14.628430,-1.0,77,FSME,LTETRS_AT,6887
HwapiExe,22883.319060,22402,18858,9.679008,1.0,52.0,FSME,LTETRS_AT,1915
bash,939.243945,710,1270,0.001402,-1.0,6.00,FSME,LTETRS_AT,40669
bm,11353.071038,6365,13827,0.339583,-1.0,16.00,FSME,LTETRS_AT,8784
find,679.527798,744,563,0.000000,-1,0,FSME,LTETRS_AT,2770
getty,757.295032,854,621,0.000000,-1,0,FSME,LTETRS_AT,8877
gpsd,5866.270207,-1,4816,0.577230,-1.0,2.00,FSME,LTETRS_AT,8586
inetd,639.287468,734,541,0.000000,-1,0,FSME,LTETRS_AT,8881
init,784.357046,888,646,2.352544,-1.0,22.00,FSME,LTETRS_AT,8884
kworker/0:1,0.000000,-1,0,0.297693,0.1,2.00,FSME,LTETRS_AT,5592
```

KUVIO 20. test.csv -tiedoston sisältö

Kuviossa 21 on Logstashin konfiguraatiotiedosto, jolla CSV-tiedosto saadaan vietyä Hadoopista Elasticsearch hakukoneen indeksiin komennolla: *$bin/logstash -f data/csv-hadoop-exec.conf.*

```
  GNU nano 2.3.1                          File: csv-hadoop-exec.conf

input {
  exec {
                command => "hadoop fs -get /demo/csv1/*.csv /home/hadoop/Data/csv/"
                interval => 10
    }
  file {
                path => "/home/hadoop/Data/csv/*.csv"
                sincedb_path => "/dev/null"
                start_position => "beginning"
  }
}

filter {
        if "COMMAND" in [message] {drop{}} #poistetaan CSV-tiedoston otsikkorivi
        if [message] == "" { drop{}} #poistetaan jos on tyhja rivi

       csv {
                separator => ","
                columns => [COMMAND,RSS_AVG,RSS_AVG_MIN,RSS_AVG_MAX,AVG_CPU,CPU_AVG_MIN,CPU_AVG_MAX,HW_TYPE,TEST_TYPE,NR_OF_MEAS]
       }
       mutate {convert => [RSS_AVG, "float"]}
       mutate {convert => [RSS_AVG_MIN, "integer"]}
       mutate {convert => [RSS_AVG_MAX, "integer"]}
       mutate {convert => [AVG_CPU, "float"]}
       mutate {convert => [CPU_AVG_MIN, "integer"]}
       mutate {convert => [CPU_AVG_MAX, "integer"]}
       mutate {convert => [NR_OF_MEAS, "integer"]}
}

output {
        stdout { codec => rubydebug }

        elasticsearch {
                hosts => "http://localhost:9200"
                index=> "csv-hadoop-exec"
                document_type => "csv"
       }
}
```

KUVIO 21. Logstash:in konfiguraatiotiedosto csv-hadoop-exec.conf

Input-osiossa käytetään exec{}-lisäosaa. Exec-lisäosan command-asetuksella voidaan ajaa komentoja. Tässä ajetaan Hadoop-komento: *hadoop fs -get /demo/csv1/*.csv /home/hadoop/Data/csv/.* Komennolla kopioidaan HDFS-tiedostojärjestelmän hakemistosta */demo/csv1/* kaikki CSV-tiedostot Linux-palvelimelle hakemistoon */home/hadoop/Data/csv/.* Tämän jälkeen File-lisäosalla tiedostot luetaan Linux-palvelimelta Logstashin Filter-osion käsiteltäväksi.

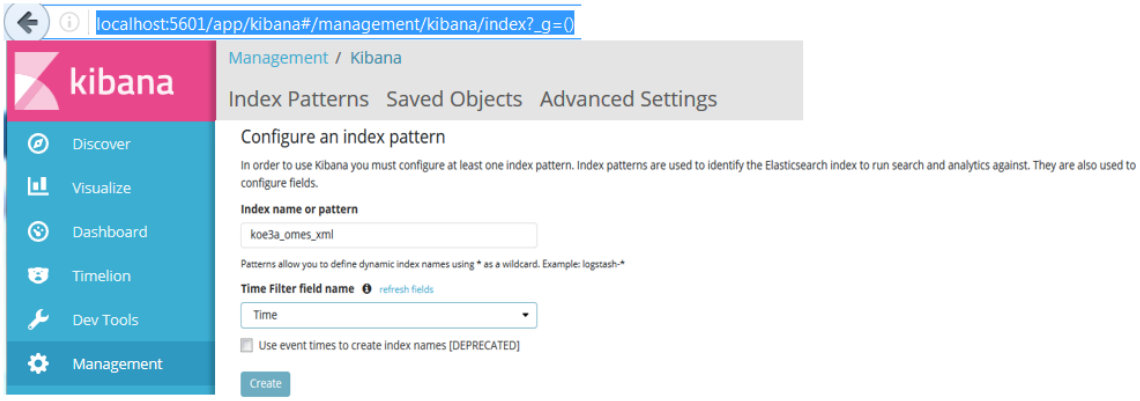
Filter-osiossa poistetaan ensin CSV-tiedoston otsikkorivi ja tyhjät rivit käsittelystä. Tämän jälkeen käytetään csv{}-lisäosaa, jossa separator-asetuksella kerrotaan mikä erotinmerkki tiedostossa on käytössä. Columns-asetuksella määritetään indeksiin tulevan dokumentin kenttien nimet. Nämä nimet vastaavat CSV-tiedoston otsikkorivillä käytettyjä sarakkeiden nimiä ja määritetään samassa järjestyksessä kuin ne ovat tiedostossa. Output-osiossa CSV-tiedoston sisältämä tieto viedään csv-hadoop-exec -nimiseen indeksiin.

5.6 Tiedon haku Kibana ohjelmassa Elasticsearch hakukoneen indeksistä

Kibana on Elacticsearch hakukoneen indeksiin tallennetun tiedon hakemiseen ja visualisointiin tarkoitettu, www-selaimessa, osoitteessa *localhost:5601* ajettava työkalu. Kibanalla on mahdollista sekä analysoida että visualisoida tietoa tekemällä erilaisia hakuja ja diagrammeja (viiva, pylväs jne.), joita voidaan viedä samaan näkymään muodostaen paneeleita (dashboards) ja joita voidaan jakaa ja upottaa www-sivuille. Kaikki visualisoinnit, tallennetut haut ja paneelit tallennetaan JSON-dokumentteina Elasticsearchin indeksiin. Kun indeksiin tulee uutta tietoa, visualisoinnit päivittyvät dynaamisesti uuden tiedon mukaan. Kibana käyttää Elasticsearchin REST (Representational State Transfer) API-rajapintaa ja käytössä ovat HTTP-protokollan CRUD (create, read, update, delete) -operaatiot. (Gupta & Gupta 2017, luku 4.)

5.6.1 Kibanan indeksimallin luonti

Kappaleen 5.4 Logstashin XML-esimerkissä luotiin ”koe3a_omes_xml” niminen indeksi. Ennenkuin tietoa voidaan hakea tästä indeksistä on luotava Kibanassa saman niminen indeksimalli (kuvio 22).

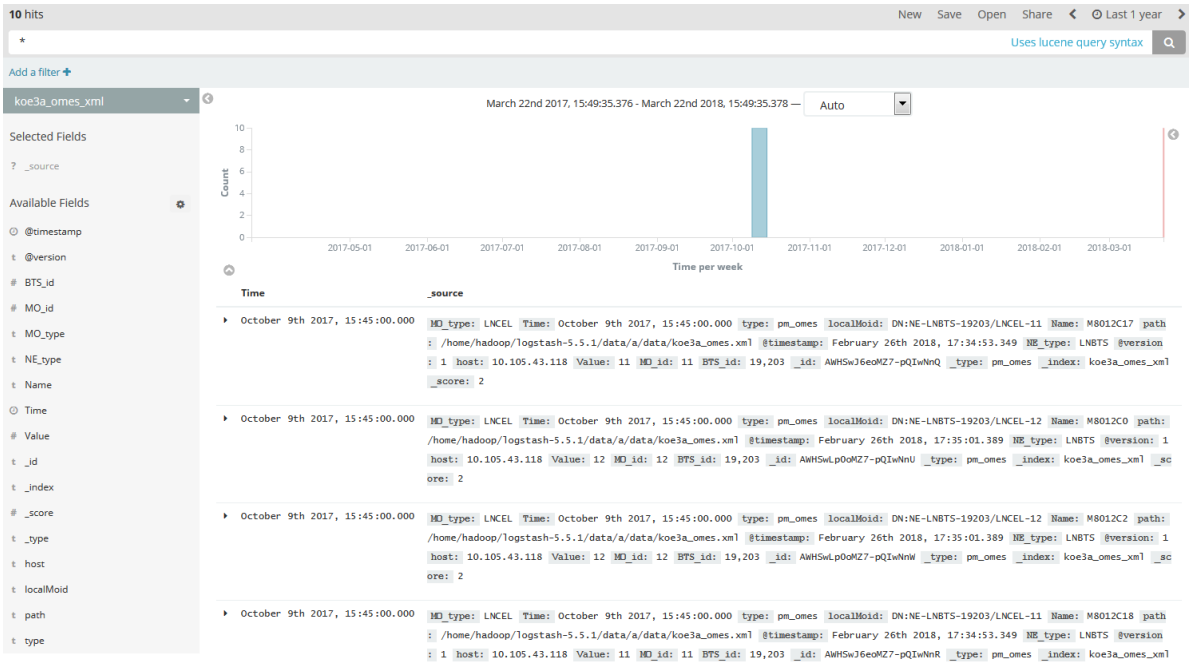


KUVIO 22. Indeksimallin luonti Kibanassa Management välilehdellä

Logstash esimerkissä luotiin Time-kenttä jokaiselle laskuriarvolle, joten valitaan se Time filter-kentän arvoksi. Toinen vaihtoehto tämän kentän arvoksi olisi @timestamp kentän arvo, joka on indeksiin kirjoitushetken aika, mutta yleensä käyttökelpoisempi tässä kohdassa on tapahtumahetken aika eikä indeksiin tallennusaika.

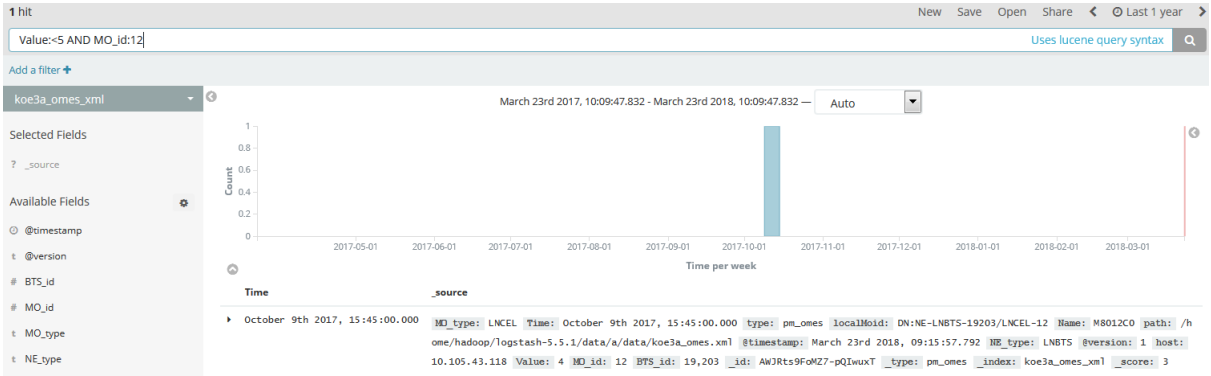
5.6.2 Tiedon hakeminen Kibanassa

Kun indeksille on luotu indeksimalli, sen jälkeen tiedolle voidaan tehdä hakuja Kibanan Discover-välilehdellä (kuvio 23). Kuvan vasemmassa yläkulmassa on indeksiin tallennettujen dokumenttien lukumäärä 10, joka vastaa Logstash-esimerkissä käytetyn XML-tiedoston laskuriarvojen määrää. Kuvassa nähdään dokumenttien lukumäärä myös visuaalisesti pylväsdiagrammilla aika-akselille sijoitettuna, mikä kertoo, että indeksin kaikki 10 merkintää tapahtuivat kaikki samana päivänä 9. lokakuuta 2017. Kuvassa näkyy myös indeksin kenttien nimet.



KUVIO 23. Näkymä koe3a_omes_xml-indeksistä Kibanan Discover-välilehdellä

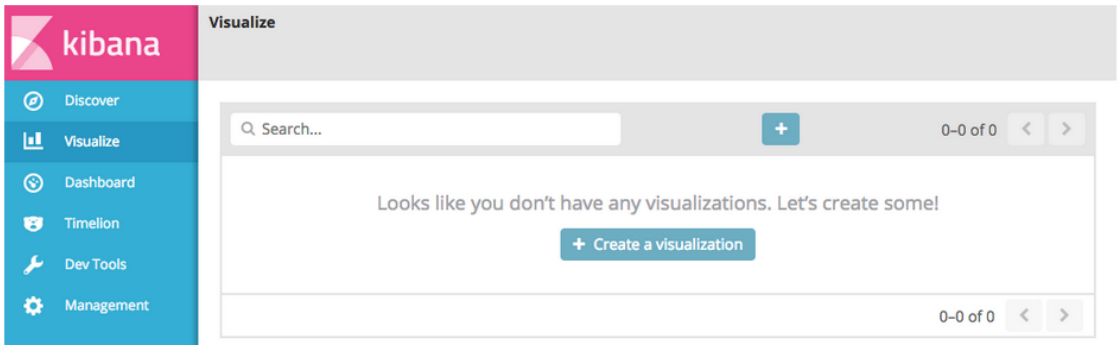
Tietoa voidaan nyt hakea indeksistä. Kun kirjoitetaan lucene query syntax -kenttään: *Value:<5 AND MO_id:12,* niin hakuehdolla löytyy indeksistä yksi dokumentti (kuvio 24).



KUVIO 24. Indeksistä löytyneet dokumentit hakuehdolla: Value:<5 AND MO_id:12

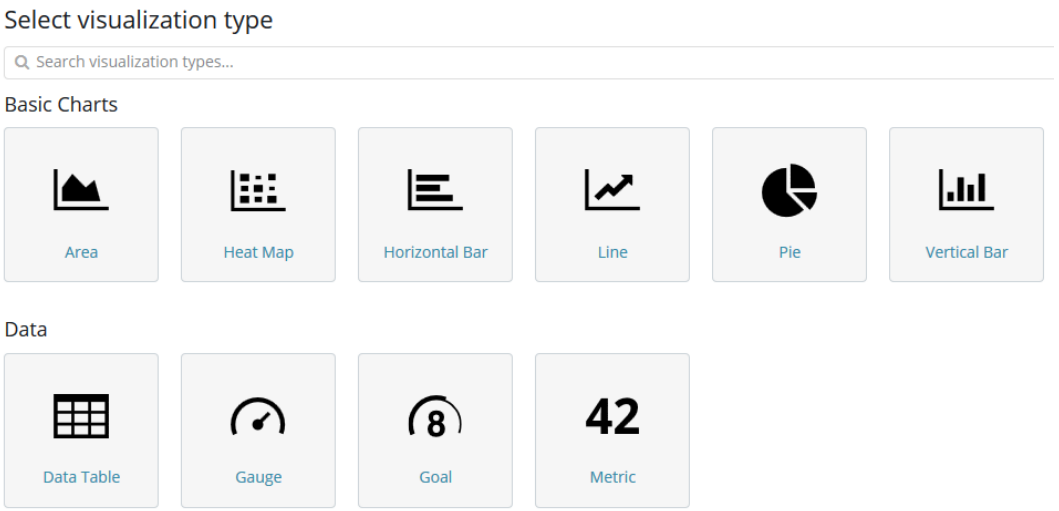
5.7 Elasticsearch indeksin visualisointi Kibanassa

Edellä nähdyn tiedon hakuesimerkin lisäksi tietoa voidaan myös visualisoida Kibanassa. Visualisoinnin tekeminen aloitetaan Kibanassa Visualize-välilehdellä Create a visualization -painikkeella (kuvio 25) (Elastic 2018k, viitattu 23.3.2018).



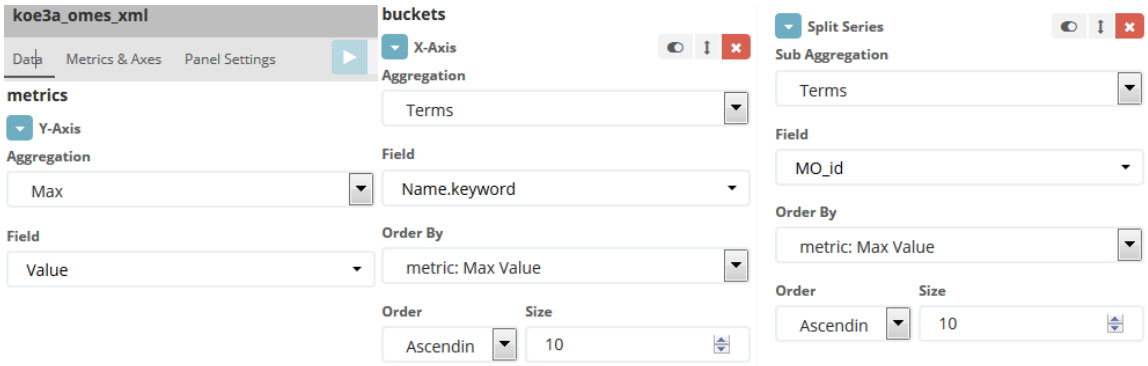
KUVIO 25. Kibanan visualisointi välilehti (sama)

Kibanasta löytyy useita visualisointi vaihtoehtoja (kuvio 26).



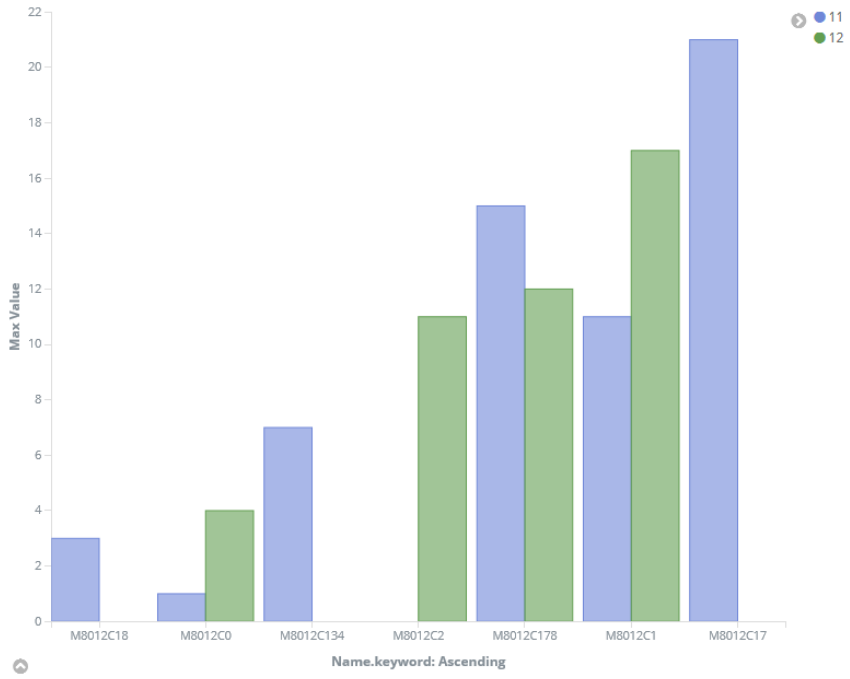
KUVIO 26. Kibanan visualisointi tyyppejä

Valitaan esimerkissä käytetylle XML-tiedostolle pylväsdiagrammi (vertical bar) visualisointi vaihtoehto. Kun visualisointi tyyppi on valittu, seuraavaksi valitaan indeksi, jolle visualisointi luodaan eli tässä tapauksessa koe3a_omes_xml. Tämän jälkeen valitaan Y- ja X-akselilla näytettävät indeksin sisältämät kentät. Valitaan Y-akselille Value-kentän sisältämät arvot ja X-akselille Name-kentän sisältämät laskureiden nimet ja kategoroidaan ne vielä Split Series-kohdassa MO_id-kentän mukaan (kuvio 27).



KUVIO 27. Pylväsdiagrammissa Y- ja X-akseleilla käytettävät kentät

Kun Y- ja X-akselilla käytettävät indeksin kentät on valittu edellä kuvatulla tavalla, saadaan koe3a_omes_xml nimisen indeksin sisältämästä tiedosta kuvion 28 mukainen pylväsdiagrammi.



KUVIO 28. Pylväsdiagrammi koe3a_omes_xml-indeksin sisältämästä tiedosta

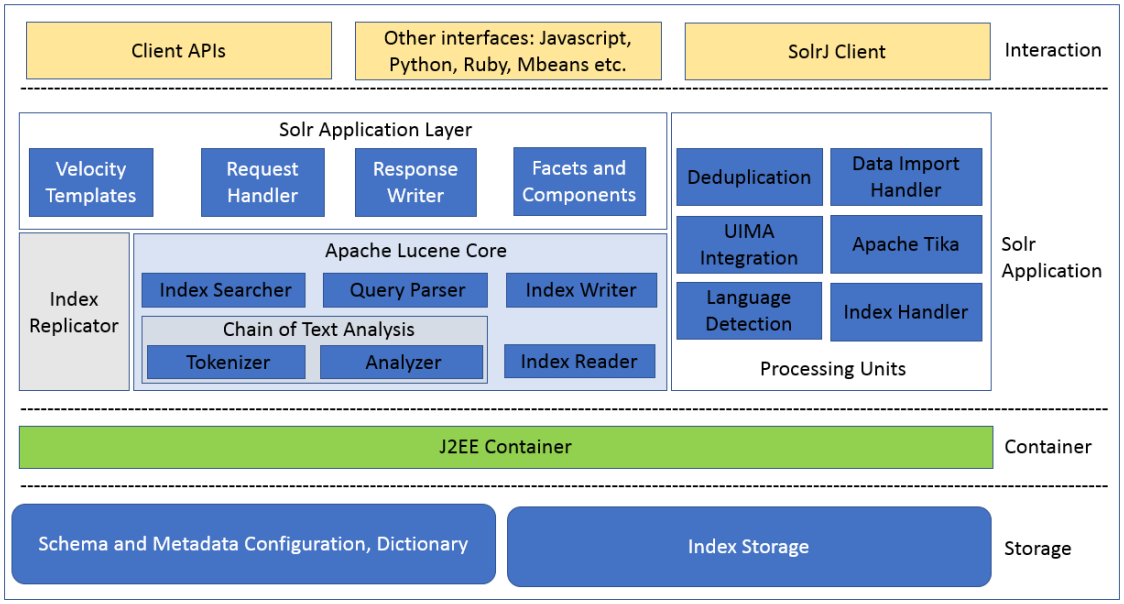
Edellä näkyvästä kuvasta nähdään mitä etuja tiedon visualisointi tuo. Tämä yksinkertainen pylväsdiagrammi kertoo monia asioita indeksin sisältämästä tiedosta. Y-akseli näyttää laskureiden arvot ja X-akseli laskureiden nimet. Siniset pylväät tarkoittavat indeksin dokumentin sisältämän MO_id-kentän arvoa 11 ja vihreät pylväät saman kentän arvoa 12. Kuvasta nähdään indeksin sisältämien kaikkien laskureiden (10kpl) arvot ja mikä on niiden suhde, jos sama laskuri esiintyy dokumentissa, jossa MO_id-kentän arvo on 11 tai 12. Kuvasta nähdään muun muassa. ensimmäisestä pylväästä, että laskuri M8012C18 esiintyy ainoastaan indeksin dokumentissa, jossa MO_id-kentän arvo on 11 ja että laskurin arvo on kolme.

6 APACHE SOLR

Apache Solr on Apache Lucene -projektin avoimen lähdekoodin hakukone palvelin, joka käyttää Java-ohjelmointikielellä kirjoitettua Apache Lucene hakukone kirjastoa (Apache Solr 2017a, viitattu 2.3.2018). Solr vaatii Javan (JRE) toimiakseen ja voidaan asentaa Linux, OS X ja Windows-koneelle (Apache Solr 2017b, viitattu 2.3.2018).

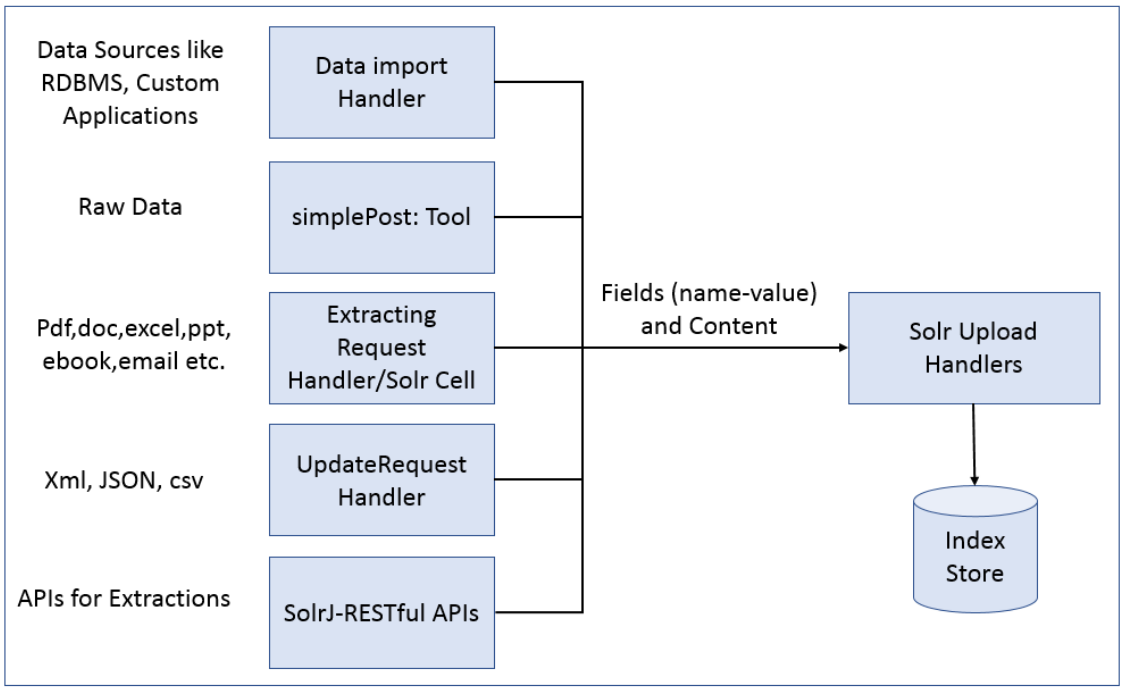
Tässä työssä ei käsitellä ohjelman asennusta ja kaikki ohjelman käyttöä koskevat komennot suoritetaan Linux-palvelimella. Työssä käytetään Apache Solr 6.6.0 versiota, joka on asennettuna Linux -palvelimelle ja jota ajetaan SolrCloud-tilassa.

Solrin toiminnallisuus voidaan jakaa neljään loogiseen tasoon (kuvio 29). Storage-taso pitää sisällään indeksi tietoa ja metadataa. Container-taso on Java alusta (J2EE), jossa ohjelmaa ajetaan. Solr hakukonetta ajetaan container-tason päällä. Ylimmäinen interaction-taso on rajapinta Solr hakukoneeseen.



KUVIO 29. Apache Solr arkkitehtuuri (Karambelkar 2014, sivu 41)

Kuviossa 30 näkyy tapoja, joilla Solrin indeksiin voi viedä tietoa.



KUVIO 30. (Karambelkar 2015, sivu 42)

6.1 Solr aloitus

Kun Solr on käynnissä palvelimella, niin ohjelman hallintaliittymä avautuu osoitteessa: http://localhost:8983/solr/#/. Solrin asennuksen tila voidaan tarkistaa antamalla ohjelman asennuskansiossa komento: *bin/solr status*. Tämä komento näyttää kaikki käynnissä olevat Solr ilmentymät palvelimella sekä perustietoa niistä, kuten tietoa versiosta, muistin käytöstä ja myös *solr_home* määrityksen sijainnin eli hakemistopolun, josta löytyy Solrin konfiguraatiotiedostoja. (Apache Solr 2017c, viitattu 2.3.2018.)

Tässä kappaleessa käsitellään ensin Solr hakukoneen käyttöä yleisesti, skeemaan liittyviä asioita ja konfiguraatiotiedostoja tilanteessa, jossa Solr on asennettu yksittäiselle palvelimelle. Kun ohjelmasta on käytössä SolrCloud versio, konfiguraatiotiedostoja hallinnoidaan ZooKeeper nimisellä ohjelmalla. SolrCloud ja ZooKeeper ohjelmia käsitellään lisää myöhemmin omassa kappaleessa.

Solr käynnistetään komennolla *bin/solr start*. Komento käynnistää ohjelmasta tausta ilmentymän portissa 8983. Käynnistyskomennosta löytyy lisää ohjeita kirjoittamalla *bin/solr start -help*. Ennen kuin indeksiin voidaan viedä tietoa, niin on luotava core. (sama.)

Solr hakukoneessa jokainen yksittäinen indeksi on core, jolla on omat konfiguraatiotiedostot: *solrconfig.xml* ja *schema.xml*. Samalla palvelimella voi olla useita indeksejä eri muotoiselle tiedolle ja jokainen indeksi muodostaa siis oman coren ja sisältää omat konfiguraatiotiedostot *schema.xml* ym. tiedostot. Ohjelmaa ajettaessa SolrCloud -tilassa indekseistä käytetään collection nimeä coren sijaan. (Apache Solr 2017d, viitattu 2.3.2018.)

Core luodaan komennolla *bin/solr create -c <coren nimi>.* Komento luo coren, joka käyttää ns. data-driven -skeemaa. Komennon kaikki käytettävissä olevat lisäasetukset löytyvät kirjoittamalla *bin/solr create -help*. (Apache Solr 2017c, viitattu 2.3.2018.)

6.2 Solr hakukoneen konfiguraatiotiedostot

Jokaisella indeksillä on indeksikohtaisia konfiguraatiotiedostoja. Nämä tiedostot sijaitsevat kotihakemistossa, jonka sijainti voi vaihdella riippuen esim. onko käytössä Solr palvelinversio vai SolrCloud versio. Kotihakemistoon tallennetaan myös indeksien sisältämä tieto. Kotihakemiston sijainnin voi tarkistaa komennolla *bin/solr status*. (Apache Solr 2017e, viitattu 9.3.2018.)

Solrin konfiguraatiotiedostoja ovat: *solr.xml*, johon määritellään Solr asennuksen käyttämiä yleisiä ominaisuuksia ja asetuksia, jotka voivat koskea useita indeksejä. Indeksikohtaisia konfiguraatiotiedostoja ovat: *core.properties, solrconfig.xml, managed-schema* (tai *schema.xml*). Core.properties -tiedostossa määritellään core kohtaisia ominaisuuksia, kuten core nimi, skeematiedoston sijainti jne. Solr palvelinversiossa indeksikohtaiset tiedostot sijaitsevat hakemistossa:

solr-home-directory/core_name1/conf/

SolrCloud version kotihakemisto on: *solr-home-directory/core_name1/* ja ei sisällä *conf/* hakemistoa, koska SolrCloud versiossa indeksikohtaisia konfiguraatiotiedostoja hallinnoidaan ZooKeeper -ohjelmalla. (sama.)

6.2.1 Schema.xml

Skeema tiedosto sisältää tiedon kentistä, mitä indeksoitavat dokumentit voivat sisältää ja miten kenttiä käsitellään, kun dokumentteja viedään Solrin indeksiin (Apache Solr Wiki 2014a, viitattu 6.3.2018). Solr hakukoneessa on käytössä oletuksena ns. Managed Schema -tila. Tässä tilassa skeema tiedoston nimi on managed-schema ja skeemaan voidaan tehdä muutoksia ainoastaan Schema API -liitännän (Application Program Interface) kautta. (Apache Solr 2017f, viitattu 6.3.2018.)

Kun käytetään managed schema -tilaa, voidaan ottaa käyttöön ns. Schemaless-tila. Tämä mahdollistaa tiedon nopean indeksoinnin ilman, että skeemaa täytyy manuaalisesti muuttaa. (Apache Solr 2017g, viitattu 6.3.2018.)

Edellä kuvattu managed-schema -tila voidaan muuttaa manuaalisesti ylläpidettävään schema.xml -tilaan seuraavasti:

1) Uudelleen nimetään tiedosto managed-schema -> schema.xml
2) Muutetaan solrconfig.xml -tiedostoa:
   a) Poistetaan rivi, jos on*:* <*schemaFactory class="ManagedIndexSchemaFactory"/>*
   b) Lisätään rivi： <*schemaFactory class="ClassicIndexSchemaFactory"/>*
3) Uudelleen ladataan core:
   HTTP -pyyntö: *admin/cores?action=RELOAD&core=core-name*

(Apache Solr 2017h, viitattu 6.3.2018.)

Oletus schema.xml -tiedostoa voidaan muuttaa omia tarpeita vastaavaksi. Skeema -tiedostossa määritellään indeksiin luettavien dokumenttien sisältämät kentät ja niiden tyypit, esim.:

<*field name="Bts_id" type="integer" indexed="true" stored="true" required="true"/>*

Staattisten kenttien lisäksi Solrissa on mahdollista luoda myös dynaamisia kenttiä. Tällöin käytetään <*dynamicField*> määrittelyä:

<*dynamicField name="*_id" type="integer" indexed="true" stored="true"*
required="true"/>

6.2.2 Solrconfig.xml

Solrconfig.xml -tiedostossa määritellään Solr hakukoneen yleisiä asetuksia. Tiedostossa määritellään mm. pyyntökäsittelijöitä (request handlers), joilla viedään dokumentteja indeksiin. Solrconfig.xml tiedosto on tallennettu solr.home/conf/ -hakemistoon. (Apache Solr 2017i, viitattu 6.3.2018.)

6.2.3 Data Import Request Handler

Data Import -pyyntökäsittelijää käytetään, kun Solr indeksiin vietävä tieto on rakenteellista XML muotoista tietoa ja käytetään XPATH-prosessoria dokumenttien tietokenttien luomiseen. Tämä käsittelijä ei ole oletuksena käytössä vaan se otetaan käyttöön lisäämällä *solrconfig.xml* -tiedostoon seuraavat rivit:

```
<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">
  <lst name="defaults">
    <str name="config">/path/to/my/DIHconfigfile.xml</str>
  </lst>
</requestHandler>
```

Edellä näkyvään config -parametriin pitää vielä määrittää käytettävän pyyntökäsittelijän konfiguraatiotiedoston sijainti. Konfiguraatiotiedostossa määritetään indeksiin luettavan tiedon lähde, noutotapa ja kuinka sitä käsitellään, jotta tiedosta saadaan muodostettua dokumenttejä indeksiin. *DIHconfigfile.xml* voi sijaita samassa hakemistossa *solrconfig.xml* -tiedoston kanssa, jolloin config -parametrin arvoksi riittää tiedostonnimi. (Apache Solr 2017j, viitattu 6.3.2018.)

Data Import -pyyntökäsittelijän käyttö

Data Import -pyyntökäsittelijä otetaan käyttöön lähettämällä selaimella Solr hakukoneeseen komentoja HTTP-pyyntöina. Kaikkien tiedostojen lataus sijainnista, joka on määritelty pyyntökäsittelijän konfiguraatiotiedostossa tapahtuu full-import -komennolla:

http://localhost:8983/solr/<solr-core-name>/dataimport?command=full-import

Delta-import -komentoa käytetään, kun halutaan viedä indeksiin vain muuttuneet tai uudet lisätyt tiedostot basedir-sijainnista:

http://localhost:8983/solr/<solr-core-name>/dataimport?command=delta-import

Muita Data Import -pyyntökäsittelijän HTTP-komentoja ovat mm.:

- http://localhost:8983/solr/<solr-core-name>/dataimport?command=reload-config
  kun konfiguraatiotiedostoa on muutettu, reload-config-komennolla muutokset tulevat voimaan ilman, että hakukonetta tarvitsee uudelleen käynnistää

- http://localhost:8983/solr/<solr-core-name>/dataimport?command=status
  komennolla nähdään tietoa indeksistä mm. lisättyjen ja poistettujen dokumenttien lukumäärä

- *http://localhost:8983/solr/<solr-core-name>/dataimport?command=show-config*
  tämä komento näyttää konfiguraatiotiedoston sisältämiä asetuksia mm. basedir-sijainti.

(sama.)

6.3 SolrCloud

Apache Solria on mahdollista käyttää SolrCloud-tilassa, jossa muodostetaan klusteri Solr palvelimista. Tämä tarjoaa hajautettua indeksointia ja hakutoimintoja ja lisää siten hakukoneen vikasietoisuutta ja myös tasaa palvelimien kuormitusta. SolrCloudin kanssa käytetään ZooKeeper nimistä ohjelmaa, jolla hallinnoidaan klusteria. (Apache Solr 2017k, viitattu 7.3.2018.)

6.3.1 Core / Collection

Kun Solr hakukonetta ajetaan yksittäisellä palvelimella, niin core vastaa indeksiä, ja kun tarvitaan useita indeksejä, niin luodaan useita coreja. SolrCloud versiossa yksittäinen indeksi voi jakautua usealle palvelimille ja tästä syystä tällaista yksittäistä loogista indeksiä kutsutaan SolrCloud versiossa nimellä collection. (Apache Solr Wiki 2014b, viitattu 7.3.2018.)

6.3.2 SolrCloud käynnistäminen

SolrCloudia voidaan ajaa myös yksittäisellä palvelimella, mutta käytännössä tuotantoympäristössä SolrCloudin käytössä on useita palvelimia tai virtuaalikoneita. SolrCloud voidaan käynnistää komennolla: *bin/solr -e cloud*, jolloin voidaan antaa perustietoja klusterin muodostamisesta mm. indeksi (collection) nimi. SolrCloud voidaan käynnistää myös oletusasetuksia käyttäen komennolla: *bin/solr -e cloud -noprompt*. (Apache Solr 2017l, viitattu 8.3.2018.)

6.3.3 SolrCloud esimerkki

Komennolla *bin/solr status* saadaan tietoa käytössä olevasta Solr asennuksesta. Kuviossa 31 nähdään, että tässä tapauksessa ohjelmasta on käytössä kaksi ilmentymää porteissa 8983 ja 7574, mikä tarkoittaa, että selaimessa Solr voidaan avata kahdessa eri osoitteessa:

http://localhost:8983/solr/#/ tai : http://localhost:7574/solr/#/

Kuviossa 31 nähdään myös ZooKeeperin osoite: localhost:9983 ja klusterissa olevien indeksien (collections) lukumäärä: 7.

```
[hadoop@hadoopnode1 solr-6.6.0]$bin/solr status

Found 2 Solr nodes:

Solr process 20748 running on port 8983
{
  "solr_home":"/home/hadoop/solr-6.6.0/example/cloud/node1/solr",
  "version":"6.6.0 5c7a7b65d2aa7ce5ec96458315c661a18b320241 - ishan - 2017-05-30
 07:32:53",
  "startTime":"2018-03-02T13:36:52.516Z",
  "uptime":"3 days, 20 hours, 20 minutes, 6 seconds",
  "memory":"103.6 MB (%21.1) of 490.7 MB",
  "cloud":{
    "ZooKeeper":"localhost:9983",
    "liveNodes":"2",
    "collections":"7"}}


Solr process 21061 running on port 7574
{
  "solr_home":"/home/hadoop/solr-6.6.0/example/cloud/node2/solr",
  "version":"6.6.0 5c7a7b65d2aa7ce5ec96458315c661a18b320241 - ishan - 2017-05-30
 07:32:53",
  "startTime":"2018-03-02T13:37:06.708Z",
  "uptime":"3 days, 20 hours, 19 minutes, 53 seconds",
  "memory":"85.4 MB (%17.4) of 490.7 MB",
  "cloud":{
    "ZooKeeper":"localhost:9983",
    "liveNodes":"2",
    "collections":"7"}}
```

KUVIO 31. Solr asennus tietoa

6.4 ZooKeeper

Apache ZooKeeper on keskitetty hallintapalvelu, jota käytetään hajautettujen sovellusten konfiguraatiotiedostojen ylläpitoon (Apache ZooKeeper 2017, viitattu 7.3.2018). Solr hakukoneen mukana tulee ZooKeeper ohjelma, mutta tuontatoympäristössä suositellaan asennettavaksi Apache ZooKeeper erikseen, koska jos Solr palvelu sammutetaan niin samalla suljetaan myös ZooKeeper, jota muut Solr ilmentymät voivat tarvita toimiakseen (Apache Solr 2017m, viitattu 8.3.2018).

Kun SolrCloud käynnistetään ensimmäisen kerran komennolla *bin/solr -e cloud,* niin Solrin konfiguraatiotiedostot ladataan automaattisesti ZooKeeper ohjelmaan ja linkitetään samassa yhteydessä luotuun indeksiin. Uusi SolrCloud indeksi voidaan luoda myös komennolla *bin/solr create -c mycollection -d _default*, jossa -c -optiossa annetaan uuden indeksin nimi ja -d -optiossa hakemiston nimi, joka sisältää käytettävät konfiguraatiotiedostot, jotka samalla ladataan ZooKeeperiin. (Apache Solr 2017n, viitattu 8.3.2018.)

SolrCloud konfiguraatiotiedostojen ylläpito ZooKeeper ohjelmalla

Kun SolrCloud konfiguraatiotiedostoja halutaan muuttaa, niin se tapahtuu seuraavasti:

1) Indeksin konfiguraatiotiedostot ladataan ZooKeeper ohjelmasta paikalliselle levylle: bin/solr zk downconfig -z <zkHost> -n <name **for** configset> -d <path to directory with configset>
esim.: *bin/solr zk downconfig -z localhost:9983 -n omes-xml -d conf/omes-xml*

2) Tehdään paikallisella levyllä oleville tiedostoille halutut muutokset.

3) Muutosten jälkeen tiedostot ladataan takaisin ZooKeeperiiin:
bin/solr zk upconfig -z <zkHost> -n <name **for** configset> -d <path to directory with configset>
esim*.: bin/solr zk upconfig -z localhost:9983 -n omes-xml -d conf/omes-xml*
(sama)

6.5 Solr esimerkki, XML-tiedoston luku indeksiin käyttämällä XSLT- ja DIH-tiedostoja

Seuraavassa esimerkissä viedään tiedosto, joka sisältää rakenteellista XML muotoista tietoa Solrin indeksiin käyttäen Data Import Handler -pyyntökäsittelijää (DIH) ja XSLT-tiedostoa. Esimerkissä käytetään samaa XML-tiedostoa, jota käytettiin kappaleessa 5.4 Logstash esimerkin yhteydessä.

Ennen kuin tietoa voidaan viedä indeksiin on luotava core tai collection. SolrCloudia käytettäessä luodaan collection komennolla: *bin/solr create -c omes-test-xml (kuvio 32).*

```
[hadoop@hadoopnode1 solr-6.6.0]$bin/solr create -c omes-test-xml

Connecting to ZooKeeper at localhost:9983 ...
INFO  - 2018-03-05 09:15:05.023; org.apache.solr.client.solrj.impl.ZkClientClust
erStateProvider; Cluster at localhost:9983 ready
Uploading /home/hadoop/solr-6.6.0/server/solr/configsets/data_driven_schema_conf
igs/conf for config omes-test-xml to ZooKeeper at localhost:9983

Creating new collection 'omes-test-xml' using command:
http://localhost:7574/solr/admin/collections?action=CREATE&name=omes-test-xml&nu
mShards=1&replicationFactor=1&maxShardsPerNode=1&collection.configName=omes-test
-xml
```

KUVIO 32. SolrCloud collection luonti

Kun collection on luotu niin se näkyy Solrissa http://localhost:8983/solr/#/ (kuvio 33).

KUVIO 33. Solr collection omes-test-xml

6.5.1 ZooKeeper / downconfig

Seuraavaksi ladataan ZooKeeper ohjelmasta juuri luodun indeksin konfiguraatiotiedostot test_conf nimiseen kansioon paikalliselle levylle:

bin/solr zk downconfig -z localhost:9983 -n omes-test-xml -d test_conf (kuvio 34)

```
[hadoop@hadoopnode1 solr-6.6.0]$bin/solr zk downconfig -z localhost:9983 -n omes-test-xml -d
test_conf

Connecting to ZooKeeper at localhost:9983 ...
Downloading configset omes-test-xml from ZooKeeper at localhost:9983 to directory /home/hadoo
p/solr-6.6.0/test_conf/conf
[hadoop@hadoopnode1 solr-6.6.0]$
```

KUVIO 34. Konfiguraatiotiedostojen lataus ZooKeeperistä hakemistoon

Edellisen komennon suorittamisen jälkeen test_conf -kansion sisältö on seuraava (kuvio 35).

```
[hadoop@hadoopnode1 solr-6.6.0]$ls test_conf/conf
currency.xml  lang            params.json    solrconfig.xml  synonyms.txt
elevate.xml   managed-schema  protwords.txt  stopwords.txt
[hadoop@hadoopnode1 solr-6.6.0]$
```

KUVIO 35. ZooKeeperistä ladatut konfiguraatiotiedostot test_conf -kansiossa

6.5.2 Schema.xml

Nimetään skeema tiedosto managed-schema-tiedosto schema.xml nimiseksi (kuvio 36).

```
  GNU nano 2.3.1                        File: managed-schema

 This is the Solr schema file. This file should be named "schema.xml" and
 should be in the conf directory under the solr home
 (i.e. ./solr/conf/schema.xml by default)
 or located where the classloader for the Solr webapp can find it.

 This example schema is the recommended starting point for users.
 It should be kept correct and concise, usable out-of-the-box.

 For more information, on how to customize this file, please see
 http://wiki.apache.org/solr/SchemaXml
```

KUVIO 36. Managed-schema -tiedosto

Lisätään schema.xml -tiedostoon XML-tiedostossa olevien kenttien nimet ja niiden tyypit (kuvio 37).

```
 GNU nano 2.3.1                                  File: schema.xml

<schema name="example-data-driven-schema" version="1.6">
<!-- **** -->
       <field name="Time" type="date" indexed="true" stored="true" required="false"/>
       <field name="localMoid" type="string" indexed="true" stored="true" required="false"/>
       <field name="Name" type="string" indexed="true" stored="true" required="false"/>
       <field name="Value" type="int" indexed="true" stored="true" required="false"/>
       <field name="NE_type" type="string" indexed="true" stored="true" required="false"/>
       <field name="MO_type" type="string" indexed="true" stored="true" required="false"/>
       <field name="BTS_id" type="int" indexed="true" stored="true" required="false"/>
       <field name="MO_id" type="int" indexed="true" stored="true" required="false"/>
<!-- **** -->
```

KUVIO 37. Schema.xml -tiedosto

Skeema tiedostosta on hyvä vielä tarkistaa, että se sisältää id-kenttä määrityksen, koska jokainen indeksiin vietävä dokumentti vaatii yksilöllisen id-avaimen ja jos XML-tiedosto ei sisällä tätä kenttää niin Solr luo sen automaattisesti (kuvio 38).

```
<field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />
<uniqueKey>id</uniqueKey>
```

KUVIO 38. Solr id-avain

6.5.3 Solrconfig.xml

Solrconfig.xml -tiedostoon Request Handlers -kohtaan lisätään Data Import -pyyntökäsittelijä, joka käyttää data-config.xml nimistä konfiguraatiotiedostoa (kuvio 39).

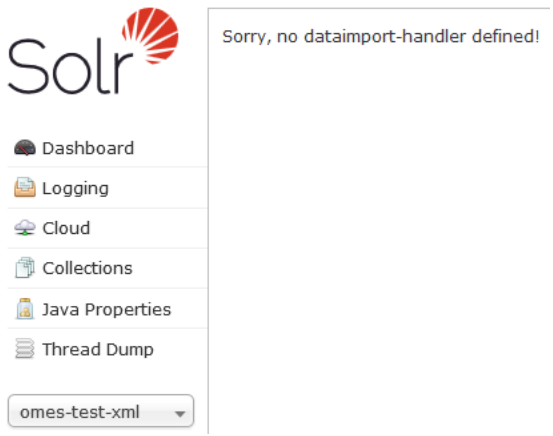
```
 GNU nano 2.3.1                                File: solrconfig.xml


<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">
<lst name="defaults">
 <str name="update.chain">add-unknown-fields-to-the-schema</str>
<str name="config">data-config.xml</str>
</lst>
</requestHandler>

<requestHandler name="/update" class="solr.UpdateRequestHandler"/>
```

KUVIO 39. Solrconfig.xml -tiedosto ja Data Import -pyyntökäsittelijä

Jos Data Import -pyyntökäsittelijää ei lisätä edellä kuvatulla tavalla niin tulee seuraava virheilmoitus (kuvio 40).



KUVIO 40. Virheilmoitus ettei Data Import -pyyntökäsittelijää ole määritelty

Jos virheilmoitus tulee edelleen niin kannattaa tarkistaa, että tiedostossa on seuraava rivi (kuvio 41).

```
 GNU nano 2.3.1                                    File: solrconfig.xml

 <lib dir="${solr.install.dir:../../../..}/dist/" regex="solr-dataimporthandler-.*\.jar" />
```

KUVIO 41. Solrconfig.xml

Koska managed-schema muutettiin schema.xml nimiseksi niin lisätään seuraava rivi (kuvio 42).

```
 GNU nano 2.3.1                                    File: solrconfig.xml

<schemaFactory class="ClassicIndexSchemaFactory"/>
```

KUVIO 42. Solrconfig.xml -tiedostossa otetaan käyttöön schema.xml

Poistetaan seuraava rivi, jos se on olemassa (kuvio 43).

```
 GNU nano 2.3.1                                    File: solrconfig.xml

<schemaFactory class="ManagedIndexSchemaFactory"/>
```

KUVIO 43. Solrconfig.xml -tiedostossa poistetaan käytöstä managed-schema

Kuviossa 44 näkyvä asetus *AddSchemaFieldsUpdateProcessorFactory* on otettu pois käytöstä, koska se on käytössä vain *ManagedIndexSchemaFactory*-asetuksen kanssa, joka edellä poistettiin

käytöstä. Tämän ollessa käytössä prosessori lisäisi skeemaan dynaamisesti kenttiä, jos sisäänluettava tiedosto sisältäisi kenttiä mitä ei ole määritetty skeemassa.

```
 GNU nano 2.3.1                                          File: solrconfig.xml

<!-- ***
    <processor class="solr.AddSchemaFieldsUpdateProcessorFactory">
      <str name="defaultFieldType">strings</str>
      <lst name="typeMapping">
        <str name="valueClass">java.lang.Boolean</str>
        <str name="fieldType">booleans</str>
      </lst>
      <lst name="typeMapping">
        <str name="valueClass">java.util.Date</str>
        <str name="fieldType">tdates</str>
      </lst>
      <lst name="typeMapping">
        <str name="valueClass">java.lang.Long</str>
        <str name="valueClass">java.lang.Integer</str>
        <str name="fieldType">tlongs</str>
      </lst>
      <lst name="typeMapping">
        <str name="valueClass">java.lang.Number</str>
        <str name="fieldType">tdoubles</str>
      </lst>
    </processor>
*** -->
```

KUVIO 44. AddSchemaFieldsUpdateProcessorFactory -asetus poistettu käytöstä

Jos kuviossa 44 näkyvää muutosta ei tee, niin voi tulla virheilmoitus:

error: “org.apache.solr.common.SolrException: This IndexSchema is not mutable.”

Schema.xml -tiedoston kohdassa käsiteltiin uniqueKey / id -avainkenttää, joka siis vaaditaan jokaiselle Solr dokumentille. Myös solrconfig.xml -tiedostossa on kohta, joka vaikuttaa tähän (kuvio 45).

```
GNU nano 2.3.1                                  File: solrconfig.xml

<updateRequestProcessorChain name="add-unknown-fields-to-the-schema">
  <!-- UUIDUpdateProcessorFactory will generate an id if none is present in the incoming document -->
  <processor class="solr.UUIDUpdateProcessorFactory">
      <str name="fieldName">id</str>
  </processor>
```

KUVIO 45. Id-avaimen luonti solrconfig.xml -tiedostossa

Jos id:n luonti ei onnistu niin voi tulla virheilmoitus:

”org.apache.solr.common.SolrException: Document is missing mandatory uniqueKey field: id”

6.5.4 Data-config.xml

Seuraavaksi tehdään tiedosto: *data-config.xml*, joka määritettiin käyttöönotettavaksi edellä Data Import -pyyntökäsitttelijän config -parametrissä (kuvio 46).

```
  GNU nano 2.3.1                                File: data-config.xml

<dataConfig>
     1) <dataSource type="FileDataSource" encoding="UTF-8" />
     2) <document name="document_name">

             3) <entity name="document"
                      4) processor="FileListEntityProcessor"
                      5) baseDir="/home/hadoop/Data/omes/xml"
                      6) fileName="."
                      7) recursive="false"
                      8) rootEntity="false"
                      9) dataSource="null">

                     10) <entity name="entity_name"
                              11) processor="XPathEntityProcessor"
                              12) url="${document.fileAbsolutePath}"
                              13) stream="true"

                              14) xsl="xslt/omes3.xsl"
                              15) useSolrAddSchema="true">
                         </entity>

              </entity>

      </document>
</dataConfig>
```

KUVIO 46. Data-config.xml -tiedosto

Data-config.xml -tiedoston sisältö selitettynä:

1) DataSource -elementissä määritetään, että tietolähteenä käytetään tekstitiedostoja
2) DataSource -elementin jälkeen tulee document -elementti, joka sisältää entity -elementtejä
3) Ulompi entity -elementti, jolla tehdään lista käsiteltävistä tiedostoista
4) Entity käyttää tiedostolistan tekemiseen FileListEntityProsessoria
5) Hakemistopolku käsiteltäviin XML-tiedostoihin
6) Filename -attribuutin arvo on säännöllinen lauseke: piste, jolla valitaan hakemiston kaikki tiedostot
7) Alihakemistoja ei käsitellä
8) Yleisesti document -elementin jälkeen välittömästi tuleva entity on ns. rootEntity, joka tuottamat dokumentit Solr indeksoi, mutta tämän entityn tuottamaa tiedostolistaa ei haluta indeksoida, joten arvoksi laitetaan false ja tämä aiheuttaa, että seuraava entity on rootEntity, jonka tuottamat dokumentit indeksoidaan
9) FileListEntityProcessor ei tarvitse tietolähdettä

10) Määritetään sisempi entity, jolla xml -tiedosto luetaan ja indeksoidaan, käyttää ulompaa entityä tietolähteenä
11) Sisempi entity käyttää XpathEntityProsessoria, jolla xml -tiedostoa käsitellään
12) Käsiteltävän XML-tiedoston sijaintipolku
13) Määritetään miten XML-tiedosto luetaan muistiin käsiteltäväksi
14) Käytettävä XSLT-tiedosto, jolla xml-tiedosto muunnetaan Solrin käyttämään xml -muotoon
15) Määritetään, että entityssä käsitelty xml-tiedosto on Solrin tukemaa xml skeema muotoa, jolla entityn tuottama dokumentti voidaan lisätä Solrin indeksiin: <add><doc><field>

(Smiley 2014, luku 4; Patil 2015, viitattu 12.3.2018; Apache Solr 2017o, viitattu 12.3.2018; Apache Solr 2017p, viitattu 12.3.2018)

6.5.5 XSLT-tiedosto

Seuraavaksi tehdään XSLT-tyylitiedosto, joka määritettiin otettavan käyttöön Data Import -pyyntö-käsittelijän konfiguraatiotiedostossa. Tehdään *omes3.xsl* niminen tiedosto *test_conf* -hakemistoon. Tyylitiedosto tulee tallentaa *conf/xslt* -hakemistoon eli tehdään tiedosto: *test_conf/conf/xslt/omes3.xsl*.

Solr-hakukoneen indeksiin viedään seuraava XML-tiedosto (kuvio 47).

```
<?xml version="1.0"?>
<OMeS xmlns="pm/cnf_lte_nsn.1.0.xsd">
<PMSetup startTime="2017-10-09T15:45:00.000+03:00:00" interval="15">
    <PMMOResult>
      <MO>
        <baseId>NE-MRBTS-19203</baseId>
        <localMoid>DN:NE-LNBTS-19203/LNCEL-11</localMoid>
      </MO>
      <MO>
        <DN>PLMN-PLMN/MCC-244/MNC-09</DN>
      </MO>
      <NE-LNBTS_1.0 measurementType="LTE_Cell_Throughput">
        <M8012C0>1</M8012C0>
        <M8012C1>11</M8012C1>
        <M8012C17>21</M8012C17>
        <M8012C18>3</M8012C18>
        <M8012C134>7</M8012C134>
        <M8012C178>15</M8012C178>
      </NE-LNBTS_1.0>
    </PMMOResult>
    <PMMOResult>
      <MO>
        <baseId>NE-MRBTS-19203</baseId>
        <localMoid>DN:NE-LNBTS-19203/LNCEL-12</localMoid>
      </MO>
      <MO>
        <DN>PLMN-PLMN/MCC-244/MNC-09</DN>
      </MO>
      <NE-LNBTS_1.0 measurementType="LTE_Cell_Throughput">
        <M8012C0>4</M8012C0>
        <M8012C1>17</M8012C1>
        <M8012C2>11</M8012C2>
        <M8012C178>12</M8012C178>
      </NE-LNBTS_1.0>
    </PMMOResult>
  </PMSetup>
</OMeS>
```

KUVIO 47. Solrin indeksiin vietävä XML-tiedosto

Kuviossa 47 näkyvä XML-tiedosto halutaan muuttaa XSLT-tiedostoa apuna käyttäen Solr hakukoneen vaatimaan muotoon (kuvio 48).

```
<?xml version="1.0" encoding="utf-8"?>
<add xmlns:x="pm/cnf_lte_nsn.1.0.xsd">
        <doc>
                <field name="Time">2017-10-09T15:45:00</field>
                <field name="localMoid">DN:NE-LNBTS-19203/LNCEL-11</field>
                <field name="Name">M8012C0</field>
                <field name="Value">11</field>
                <field name="NE_type">LNBTS</field>
                <field name="BTS_id">19203</field>
                <field name="MO_type">LNCEL</field>
                <field name="MO_id">11</field>
        </doc>
        <doc>
                <field name="Time">2017-10-09T15:45:00</field>
                <field name="localMoid">DN:NE-LNBTS-19203/LNCEL-11</field>
                <field name="Name">M8012C1</field>
                <field name="Value">11</field>
                <field name="NE_type">LNBTS</field>
                <field name="BTS_id">19203</field>
                <field name="MO_type">LNCEL</field>
                <field name="MO_id">11</field>
        </doc>
        <doc>
                <field name="Time">2017-10-09T15:45:00</field>
                <field name="localMoid">DN:NE-LNBTS-19203/LNCEL-11</field>
                <field name="Name">M8012C17</field>
                <field name="Value">11</field>
                <field name="NE_type">LNBTS</field>
                <field name="BTS_id">19203</field>
                <field name="MO_type">LNCEL</field>
                <field name="MO_id">11</field>
        </doc>
```

KUVIO 48. Solrin indeksiin vietävä XML-tiedosto XSLT-muunnoksen jälkeen

XML-tiedoston muunnos Solr hakukoneen vaatimaan XML-muotoon saadaan aikaan seuraavalla XSLT-tiedostolla (kuvio 49).

```
 GNU nano 2.3.1                                File: omes3.xsl

<xsl:stylesheet version="1.0"
            1) xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            2) xmlns:x="pm/cnf_lte_nsn.1.0.xsd">
 <xsl:output method="xml" indent="yes"/> 3)
 4)<xsl:template match="/x:OMeS/x:PMSetup">
 5) <add>
      6) <xsl:for-each select="x:PMMOResult">
            7) <xsl:variable name="localmoid">
                    <xsl:value-of select="x:MO/x:localMoid/text()"/>
               </xsl:variable>
         8)  <xsl:for-each select="x:NE-LNBTS_1.0/*">
              9) <doc>
                 10) <field name="Time">
                        <xsl:value-of select="substring-before(//@startTime,'.')"/>
                    </field>
                 11) <field name="localMoid">
                            <xsl:value-of select="$localmoid"/>
                    </field>
                 12) <field name="Name">
                        <xsl:value-of select="name()"/>
                    </field>
                 13) <field name="Value">
                         <xsl:value-of select="text()"/>
                    </field>
                  14) <field name="NE_type">
                        <xsl:value-of select="substring-before(substring-after($localmoid,'-'),'-')"/>
                    </field>
                 15) <field name="BTS_id">
                        <xsl:value-of select="substring-after(substring-after(substring-before($localmoid,'/'),'-'),'-')"/>
                    </field>
                 16) <field name="MO_type">
                        <xsl:value-of select="substring-before(substring-after($localmoid,'/'),'-')"/>
                    17) </field>
                      <field name="MO_id">
                        <xsl:value-of select="substring-after(substring-after($localmoid,'/'),'-')"/>
                    </field>
                  </doc>
              </xsl:for-each>
       </xsl:for-each>
    </add>
   </xsl:template>
</xsl:stylesheet>
```

KUVIO 49. XSLT-tiedosto

Kuvion 49 XSLT-tiedosto selitettynä:

1) Määritetään XSLT-nimiavaruus, jotta voidaan käyttää dokumentissa käyttöön otettuja XSLT-elementtejä
2) XML-tiedostossa Omes -elementtiin on määritelty nimiavaruus (xml name space) <OMeS xmlns="pm/cnf_lte_nsn.1.0.xsd">, jotta nimiavaruutta voidaan hyödyntää, määritetään se otettavaksi käyttöön kirjaimella x
3) Määritetään, että tulostettava dokumentti on XML-muotoista ja sisennetty hierarkisen rakenteen mukaan
4) Määritetään sääntö, joka kohdistuu XML-tiedostossa XML-elementtiin <OmeS><PMSetup> käyttäen edellä määritettyä nimiavaruutta /x:
5) Jotta indeksiin vietävä dokumentti on Solrin vaatimassa muodossa <add><doc><field> niin lisätään <add> -elementti
6) Määritetään silmukka, jolla käydään läpi PMMOResult elementtiä
7) Määritetään localmoid niminen muuttuja, jonka arvoksi tulee localMoid -elementin sisältö
8) Määritetään silmukka, jolla käydään läpi NE-LNBTS_1.0 -elementin kaikki lapsielementit, joiden määrä voi vaihdella
9) <doc> -elementillä määritetään indeksiin vietävä dokumentti
10) Määritetään Time-kenttä, jonka arvoksi tulee startTime-attribuutin sisältö merkkijonossa olevaan ensimmäiseen pisteeseen saakka
11) localMoid -kentän arvoksi tulee localmoid -muuttujan sisältö
12) Name -kentän arvoksi tulee käsiteltävän lapsielementin nimi
13) Value -kentän arvoksi tulee vastaavan lapsielementin sisältö
14) NE_type kentän arvoksi luetaan localmoid -muuttujasta ensin väliviivan jälkeinen merkkijono kokonaisuudessaan ja sen jälkeen väliviivaa edeltävä merkkijono, jolloin kentän arvoksi tulee esim. LNBTS
15) – 17) Kentät BTS_id, MO_type ja MO_id muodostetaan vastaavalla muuttujan localmoid sisällöstä, kuten kohdassa 14

(Datatypic 2014, viitattu 12.3.2018; w3schools 2018d, viitattu 9.2.2018)

XSLT-tiedoston toimivuutta XML-tiedoston kanssa voi testata myös internetistä löytyvillä työkaluilla, kuten esim. Xmlper (Xmlper 2018, viitattu 11.4.2018).

6.5.6 ZooKeeper / upconfig

Kun konfiguraatiotiedostot on saatu tehtyä edellä kuvatulla tavalla, sen jälkeen ne voidaan ladata takaisin ZooKeeper ohjelmaan, jolloin uudet konfiguraatiotiedostot ovat Solr hakukoneen käytössä.

Tiedostojen lataus paikalliselta levyltä ZooKeeper ohjelmaan tapahtuu komennolla:
bin/solr zk upconfig -z localhost:9983 -n omes-test-xml -d test_conf

Asetukset voi vielä ladata ZooKeeper ohjelmasta uuteen hakemistoon paikalliselle levylle ja tarkistaa tällä tavoin, että tiedostot ovat oikeita ja sisältävät tehdyt muutokset:
bin/solr zk downconfig -z localhost:9983 -n omes-test-xml -d test_conf1

Jos esimerkiksi *managed-schema*-tiedosto edelleen näkyy niin sen voi poistaa Zookeeper ohjelmasta komennolla: *bin/solr zk rm /configs/omes-test-xml/managed-schema -z localhost:9983.*

ZooKeeper komennosta löytyy ohjeita komennolla: *bin/solr zk -help*

6.5.7 Esimerkki XML-tiedoston luku indeksiin

Uudet konfiguraatiotiedostot otetaan ensin käyttöön komennolla:

http://localhost:8983/solr/admin/collections?action=RELOAD&name=omes-test-xml

Tämän jälkeen tietoa voidaan viedä indeksiin komennolla:
http://localhost:8983/solr/omes-test-xml/dataimport?command=full-import&clean=false&commit=true

Tämä komento vie tietoa sijainnista, joka määriteltiin *data-config.xml* -tiedostossa. Kuviossa 50 nähdään, että indeksiin lisättiin 10 dokumenttia.

KUVIO 50. Indeksiin lisätty 10 dokumenttia full-import komennolla

XML-tiedoston jokaisesta laskuriarvosta muodostetaan siis oma dokumentti indeksiin (kuvio 51).

KUVIO 51. Indeksin dokumentti

Esimerkki XML-tiedosto *koe3a_omes.xml* voitaisiin viedä indeksiin myös komennolla:

curl "http://localhost:8983/solr/omes-test-xml/update?commit=true&tr=omes3.xsl" -H "Content-Type: text/xml" --data-binary @koe3a_omes.xml

Edellisessä komennossa tr-parametri kohdassa määritetään käytettävä XSLT-tiedosto, jolla XML-tiedosto muunnetaan toiseen muotoon ja indeksoitava tiedosto on komennon lopussa @-merkin jälkeen.

6.6 Solr esimerkki, CSV-tiedoston luku indeksiin Hadoop tiedostojärjestelmästä

Esimerkissä käytetään samaa CSV-tiedostoa kuin mitä Logstash esimerkin yhteydessä käytettiin . Kuviossa 52 näkyy CSV-tiedoston sijainti Hadoop tiedostojärjestelmässä.

```
[hadoop@hadoopnode1 logstash-5.5.1]$hadoop fs -ls /demo/csv1
Found 1 items
-rw-r--r--   1 hadoop supergroup        740 2018-03-27 17:54 /demo/csv1/test.csv
[hadoop@hadoopnode1 logstash-5.5.1]$
```

KUVIO 52. test.csv -tiedosto Hadoop HDFS-tiedostojärjestelmässä

Luodaan ensin uusi *test-csv* niminen indeksi luettavalle CSV-tiedostolle komennolla: *bin/solr create -c test-csv* ja tämän jälkeen CSV-tiedosto voidaan viedä indeksiin.

Kun *test-csv*-indeksi edellä luotiin, niin käytössä on oletus skeema: *Managed Schema.* Kappaleen 6.5 esimerkissä indeksoitiin XML-tiedosto Solr hakukoneeseen, jolloin *Managed Schema* vaihdettiin schema.xml -tiedostoon, mutta tätä ei siis tarvitse tehdä CSV-tiedoston kanssa vaan käytetään oletus skeemaa. Kun CSV muotoista tietoa viedään indeksiin, komennoissa kerrotaan vietävän tiedon muoto, jonka mukaan Solr osaa käsitellä tietoa: Content-Type: application/csv tai Content-Type: text/csv (ks. indeksointi komennot alla). Esimerkissä *Managed Scheman* kanssa hyödynnetään myös ns. Schemaless Mode -tilaa, jolla dokumentin kentät ja niiden tietotyypit pyritään määrittämään automaattisesti. (Apache Solr 2017g, viitattu 6.3.2018; Apache Solr 2017h, viitattu 6.3.2018; Apache Solr 2017o, viitattu 12.3.2018.)

Indeksin luonnin jälkeen kopioidaan CSV-tiedosto Hadoopista paikalliselle Linux-palvelimelle komennolla:*curl -i -L -o test.csv "http://localhost:50070/webhdfs/v1/demo/csv1/test.csv?op=OPEN"* jonka jälkeen tiedosto indeksoidaan Solriin komennolla: *curl 'http://localhost:8983/solr/test-csv/update?commit=true' --data-binary @test.csv -H 'Content-type:application/csv'*

tai komennolla: *bin/post -c test-csv -type text/csv test.csv.*

Kun indeksointi komentoa ajetaan ensimmäistä kertaa, tiedoston toisen rivin kohdalla tulee virheilmoitus: “*Error adding field 'CPU_AVG_MAX'='52.0' msg=For input string: "52.0"</str><int name="code">400</int></lst>”*

Tämä johtuu siitä, että ensimmäisenä indeksiin viedyn HWR-rivin CPU_AVG_Max arvo on kokonaisluku 77 ja seuraavalla rivillä ko. kentän arvo on 52.0, joka ei ole enää INT (integer) muotoa (kuvio 53).

	A	B	C	D	E	F	G	H	I	J
1	COMMAND	RSS_AVG	RSS_AVG_MIN	RSS_AVG_MAX	AVG_CPU	CPU_AVG_MIN	CPU_AVG_MAX	HW_TYPE	TEST_TYPE	NR_OF_MEAS
2	HWR	22354.135618	14352	18704	14.628430	-1.0	77	FSME	LTETRS_AT	6887
3	HwapiExe	22883.319060	22402	18858	9.679008	1.0	52.0	FSME	LTETRS_AT	1915
4	bash	939.243945	710	1270	0.001402	-1.0	6.00	FSME	LTETRS_AT	40669
5	bm	11353.071038	6365	13827	0.339583	-1.0	16.00	FSME	LTETRS_AT	8784
6	find	679.527798	744	563	0.000000	-1	0	FSME	LTETRS_AT	2770
7	getty	757.295032	854	621	0.000000	-1	0	FSME	LTETRS_AT	8877
8	gpsd	5866.270207	-1	4816	0.577230	-1.0	2.00	FSME	LTETRS_AT	8586
9	inetd	639.287468	734	541	0.000000	-1	0	FSME	LTETRS_AT	8881
10	init	784.357046	888	646	2.352544	-1.0	22.00	FSME	LTETRS_AT	8884
11	kworker/0:1	0.000000	-1	0	0.297693	0.1	2.00	FSME	LTETRS_AT	5592

KUVIO 53. CSV-tiedoston sisältö

Kun käytössä on Managed Schema, skeemaan voidaan tehdä muutoksia HTTP API:n kautta. API:n kautta voidaan tarkistaa kenttien tietotyypit:

komento: curl http://localhost:8983/solr/test-csv/schema/fields/CPU_AVG_MAX?wt=json

{"responseHeader":{"status":0,"QTime":0},"field":{"name":"CPU_AVG_MAX","type":"tlongs"}}

ja komento: *curl http://localhost:8983/solr/test-csv/schema/fields/CPU_AVG_MIN?wt=json*

{"responseHeader":{"status":0,"QTime":0},"field":{"name":"CPU_AVG_MIN","type":"tdoubles"}}

Muutetaan CPU_AVG_MAX-kentän tietotyyppi samaksi kuin CPU_AVG_MIN, tlongs -> tdoubles:

```
curl -X POST -H 'Content-type:application/json' --data-binary '{
  "replace-field":{
    "name":"CPU_AVG_MAX",
    "type":"tdoubles"}
}' http://localhost:8983/solr/test-csv/schema
```

(Apache Solr 2017q, viitattu 3.4.2018 .)

Viedään CSV-tiedosto uudelleen indeksiin:

curl 'http://localhost:8983/solr/test-csv/update?commit=true' --data-binary @test.csv -H 'Content-type:application/csv'

ja nyt CSV-tiedoston rivit näkyvät dokumentteina Solrin indeksissä (kuvio 54).


```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":2,
    "params":{
      "q":"*:*",
      "indent":"on",
      "wt":"json",
      "_":"1522328313558"}},
  "response":{"numFound":10,"start":0,"docs":[
      {
        "COMMAND":["HWR"],
        "RSS_AVG":[22354.135618],
        "RSS_AVG_MIN":[14352],
        "RSS_AVG_MAX":[18704],
        "AVG_CPU":[14.62843],
        "CPU_AVG_MIN":[-1.0],
        "CPU_AVG_MAX":[77.0],
        "HW_TYPE":["FSME"],
        "TEST_TYPE":["LTETRS_AI"],
        "NR_OF_MEAS":[6887],
        "id":"48051836-c4b1-4426-b412-7522532c3301",
        "_version_":1596277096366211072},
      {
        "COMMAND":["HwapiExe"],
        "RSS_AVG":[22883.31906],
        "RSS_AVG_MIN":[22402],
        "RSS_AVG_MAX":[18858],
        "AVG_CPU":[9.679008],
        "CPU_AVG_MIN":[1.0],
        "CPU_AVG_MAX":[52.0],
        "HW_TYPE":["FSME"],
        "TEST_TYPE":["LTETRS_AI"],
        "NR_OF_MEAS":[1915],
        "id":"73c83711-0090-48a8-9b26-e1b689a120e1",
        "_version_":1596277096374599680},
      {
```

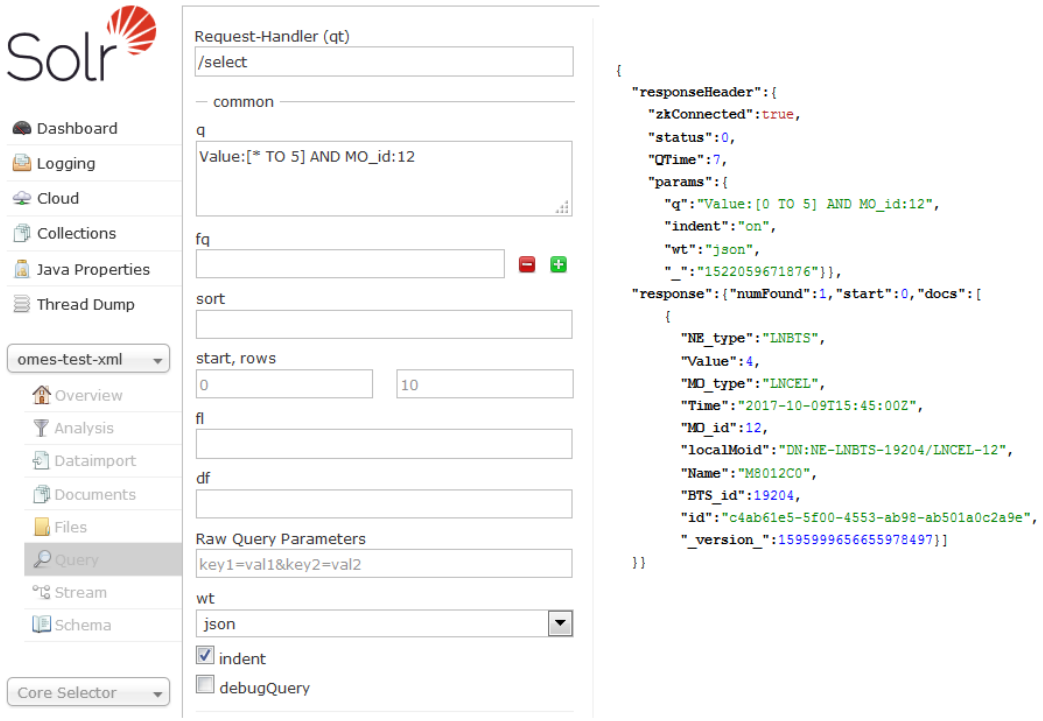

KUVIO 54. CSV-tiedosto indeksoituna Solrin dokumenteiksi

CSV-esimerkissä muutettiin siis ainoastaan yhden kentän tietotyyppiä Schema API:n kautta eikä muita muutoksia tehty konfiguraatiotiedostoihin, joten ZooKeeper-ohjelmaa ei tässä esimerkissä tarvinnut käyttää niinkuin XML-esimerkin yhteydessä tehtiin.

6.7 Tiedon hakeminen Solr hakukoneen indeksistä

Koska Solr pohjautuu Lucene hakukonekirjaston käyttöön, niin Solr hakukoneessa käytetään Lucene kyselykieltä. Edistyneemmissä Solr hakukoneen kyselyissä voi olla eroavaisuuksia verrattuna Lucene kyselykieleen. (Serafini 2013, sivu 137.)

Elasticsearch hakukoneen tiedonhaku esimerkissä indeksiin tehtiin kysely*: Value:<5 AND MO_id:12* ja löydettiin näillä ehdoilla yksi dokumentti. Tehdään vastaava haku Solr hakukoneeseen, mikä Solr hakukoneessa on muotoa: *Value:[* TO 5] AND MO_id:12* (Value ja MO_id ovat indeksissä olevia kenttien nimiä) ja huomataan, että näillä ehdoilla myös Solr hakukoneessa löytyy ainoastaan yksi dokumentti (kuvio 55).



KUVIO 55. Haku Solr:in indeksistä

Kuviossa 55 näkyy Solr hakukoneen kyselykenttä osoitteessa: localhost:8983/solr/#/omes-test-xml/query. Vastaava kysely voidaan tehdä myös HTTP-komennolla selaimessa:

localhost:8983/solr/omes-test-xml/select?indent=on&q=Value:[* TO 5] AND MO_id:12&wt=json

jolloin selain ikkunaan tulostuu kuvion 56 näkymä.

```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":1,
    "params":{
      "q":"Value:[0 TO 5] AND MO_id:12",
      "indent":"on",
      "wt":"json"}},
  "response":{"numFound":1,"start":0,"docs":[
      {
        "NE_type":"LNBTS",
        "Value":4,
        "MO_type":"LNCEL",
        "Time":"2017-10-09T15:45:00Z",
        "MO_id":12,
        "localMoid":"DN:NE-LNBTS-19204/LNCEL-12",
        "Name":"M8012C0",
        "BTS_id":19204,
        "id":"c4ab61e5-5f00-4553-ab98-ab501a0c2a9e",
        "_version_":1595999656655978497}]
  }}
```


KUVIO 56. Kysely q=Value:[TO 5] AND MO_id:12&wt=json selaimessa*

Edelliseen HTTP-komentoon voidaan lisätä myös tulostettavat kentät Name ja BTS_id:

http://localhost:8983/solr/omes-test-xml/select?indent=on&q=Value:[0%20TO%205]%20AND%20MO_id:12&fl=Name+BTS_id&wt=json

jolloin näytölle tulostuu ainoastaan dokumentin näiden kenttien sisältämä tieto (kuvio 57).


```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":1,
    "params":{
      "q":"Value:[0 TO 5] AND MO_id:12",
      "indent":"on",
      "fl":"Name BTS_id",
      "wt":"json"}},
  "response":{"numFound":1,"start":0,"docs":[
      {
        "Name":"M8012C0",
        "BTS_id":19204}]
  }}
```


KUVIO 57. Kysely q=Value:[TO 5] AND MO_id:12&fl=Name+BTS_id&wt=json selaimessa*

HTTP-komentoja voidaan kirjoittaa myös Linux:ssa:

1) *curl -X POST 'http://localhost:8983/solr/omes-test-xml/select?indent=on&wt=json' --data-urlencode 'q=Value:[* TO 5] AND MO_id:12'*

2) *curl -X GET 'http://localhost:8983/solr/omes-test-xml/select?q=Value%3A%5B*%20TO%205%5D%20AND%20MO_id%3A12&wt=json'*

Edellisestä esimerkistä nähdään, että kohdan 1 cURL POST -komennossa parametriä –data-urlencode käytettäessä voidaan kirjoittaa selväkielisenä HTTP-komennossa käytettäviä kenttiä ja arvoja, kun taas kohdan 2 GET-komennossa ne ovat URL-koodauksen vaatimassa muodossa. (sama.)

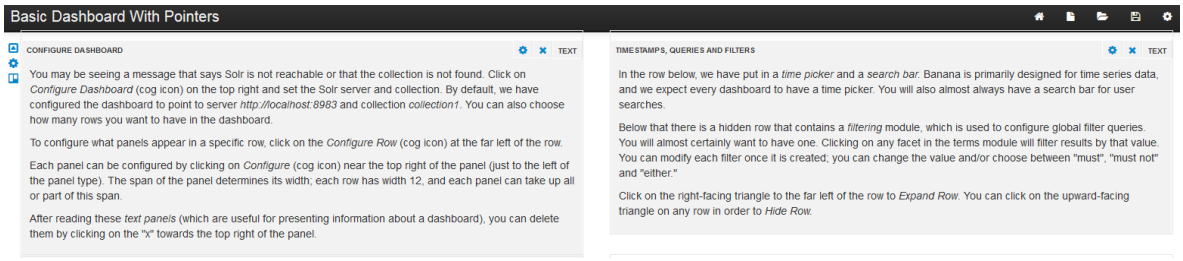
6.8 Solr hakukoneen indeksin visualisointi Banana ohjelmalla

Banana kehitysprojekti on Kibana ohjelmasta eriytynyt visualisointiohjelma, jolla voidaan visualisoida Solr hakukoneeseen tallennettua tietoa (Github 2018, viitattu 5.4.2018).

Visualisoidaan XML-tiedoston sisältämä tieto, joka vietiin Solr hakukoneen indeksiin kappaleessa 6.5. Selvitetään ensin visualisoitavan indeksin tiedot komennolla: *bin/solr healthcheck -c omes-test-xml.* Alla olevassa tulosteessa URL-kohdassa nähdään, että *omes-test-xml* indeksi löytyy Solr ilmentymästä, jota ajetaan portissa 7574.

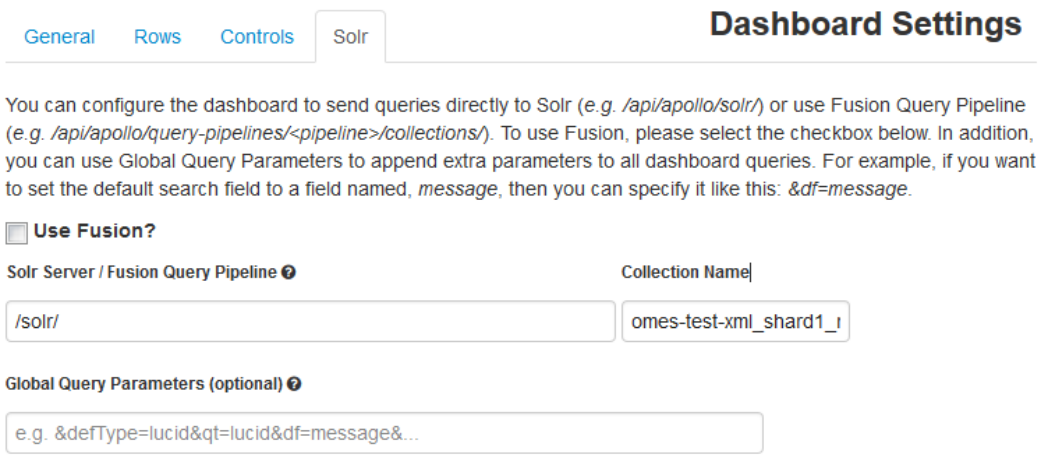

```
{
  "collection":"omes-test-xml",
  "status":"healthy",
  "numDocs":10,
  "numShards":1,
  "shards":[{
     "shard":"shard1",
     "status":"healthy",
     "replicas":[{
        "name":"core_node1",
        "url":"http://localhost:7574/solr/omes-test-xml_shard1_replica1/",
        "numDocs":10,
        "status":"active",
        "uptime":"32 days, 20 hours, 27 minutes, 15 seconds",
        "memory":"64.4 MB (%13.1) of 490.7 MB",
        "leader":true}]}]}
```

Edellä selvitettiin, että indeksin käytössä on portti 7574. Tästä syystä Banana ohjelma avataan myös tässä portissa: http://localhost:7574/solr/banana-release/src/index.html#/dashboard. Kun Banana avataan niin näkyy oletusnäkymä, koska mitään visualisointia ei ole vielä tehty. Visualisoinnin tekeminen aloitetaan klikkaamalla hammasratasta ikkunan oikeassa yläkulmassa (kuvio 58).



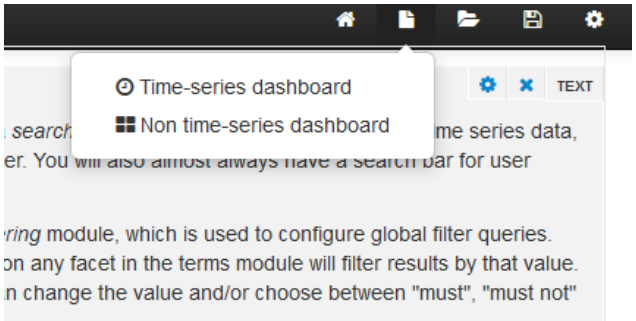
KUVIO 58. Banana aloitusnäkymä

Tämän jälkeen Solr-välilehdellä Collection Name -kohtaan kirjoitetaan indeksin nimi (kuvio 59).



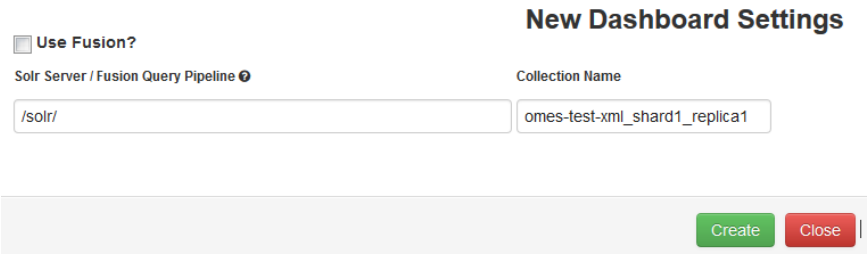
KUVIO 59. Visualisoitavan indeksin nimi Collection Name-kentässä

Seuraavaksi valitaan New ja Non time-series dashboard (kuvio 60).



KUVIO 60. Uusi dashboard

Avautuvaan ikkunaan Collection Name -kenttään kirjoitetaan indeksin nimi ja klikataan Create (kuvio 61).

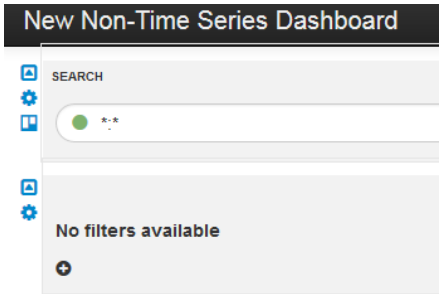


KUVIO 61. Dashboard asetukset

Tämän jälkeen näkyy mm. indeksin dokumenttien lukumäärä sekä dokumenteissa olevat kentät (kuvio 62).

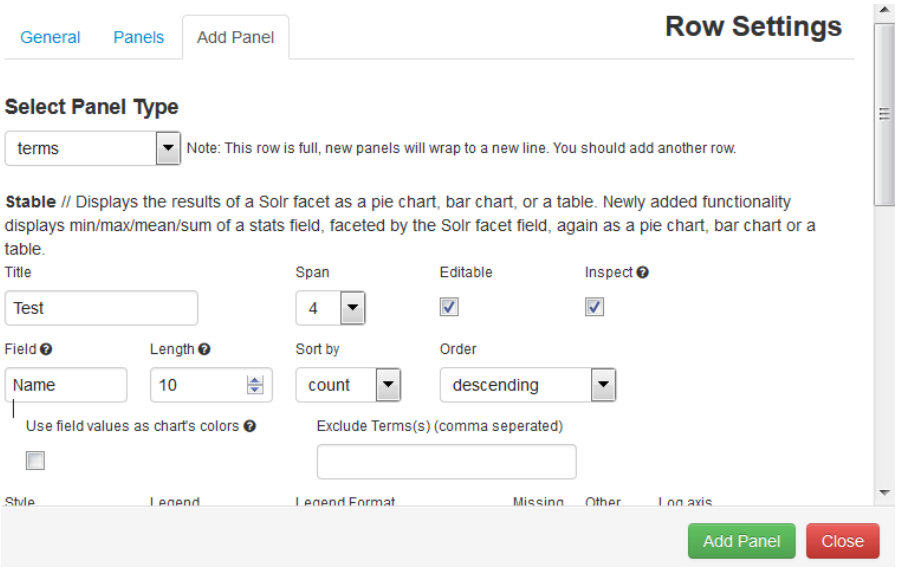
KUVIO 62. Omes-test-xml -indeksin sisältö Banana ohjelmassa

Luodaan indeksin sisältämälle tiedolle pylväsdiagrammi havainnollistamaan tietoa visuaalisesti. Pylväsdiagrammin tekeminen aloitetaan klikkaamalla hammasrattaan kuvaketta ikkunan vasemmassa yläkulmassa (kuvio 63).



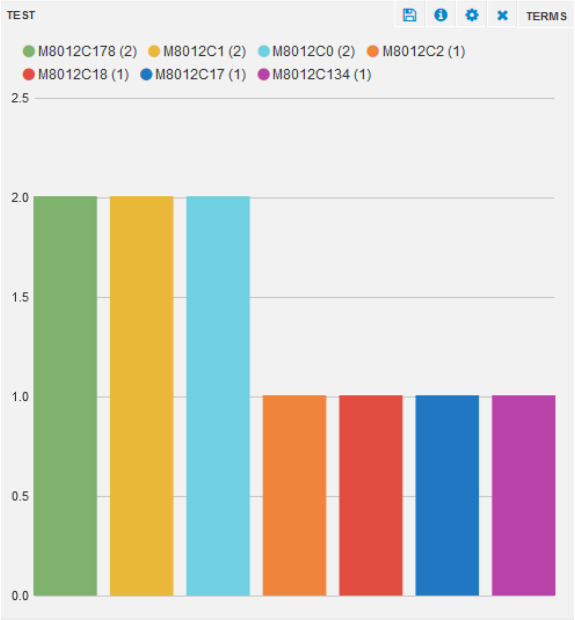
KUVIO 63. Pylväsdiagrammin aloitus hammasratas-kuvakkeella

Add Panel -välilehdellä valitaan Select Panel Type -kenttään term-vaihtoehto. Title-kentässä voidaan antaa pylväsdiagrammille nimi ja Field-kentässä valitaan dokumentin kenttä, jonka tietoja pylväsdiagrammilla näytetään eli valitaan tähän kohtaan Name, joka sisältää laskurin nimen (kuvio 64).



KUVIO 64. Pylväsdiagrammin asetuksia

Kun pylväsdiagrammin asetukset on valittu ja klikataan Add Panel -painiketta, muodostuu kuvion 65 mukainen pylväsdiagrammi, josta nähdään laskureiden nimet ja niiden esiintymiskerrat indeksissä.



KUVIO 65. Pylväsdiagrammi

7 TULOKSET JA JOHTOPÄÄTÖKSET

Opinnäytetyön tavoitteena oli selvittää miten tukiasematestauksen tuottamaa lokitietoa voidaaan käsitellä avoimen lähdekoodin hakukoneilla. Testattaviksi hakukoneiksi valittiin Elasticsearch ja Apache Solr. Molempien työssä käsiteltyjen hakukoneiden kohdalla esitettiin käytännön esimerkit kahden tietomuodoltaan eri tyyppisen lokitiedon viemisestä hakukoneen indeksiin. Ensin käsiteltiin XML- ja CSV-tiedoston vienti Elasticsearch hakukoneen indeksiin ja sen jälkeen samat tiedostot vietiin Apache Solr hakukoneeseen. Nämä kaksi tietomuodoltaan erilaista lokitiedostoa vietiin onnistuneesti molempien hakukoneiden indekseihin, jonka jälkeen tiedolle voitiin tehdä hakuja ja sitä voitiin esittää visuaalisessa muodossa.

Tiedon oikeellisuus

Tietoa vietäessä hakukoneen indeksiin on huolehdittava siitä, että indeksoitu tieto on oikeaa, eikä siinä ole virheitä. Tiedosto on mahdollista lukea indeksiin useaan kertaan, jos tätä ei huomiota. Elasticsearch hakukoneen XML-esimerkissä tähän viitattiin lyhyesti kappaleessa 5.4, kun käsiteltiin Logstash konfiguraatiotiedoston input-osion file-lisäosaa. File-lisäosassa on sincedb_path -asetus, joka pitää yllä sisäistä tietokantaa indeksiin viedyistä tiedoista. Eli kun asetuksen ottaa käyttöön, niin Logstash osaa viedä indeksiin file-lisäosan path-kohdassa määritellystä hakemistosta aina edellisen ajon jälkeiset uudet tiedostot tai muuttuneiden tiedostojen uudet rivit. Solr hakukoneessa tämä ominaisuus toteutetaan Delta import -komennolla, joka mainittiin kappaleessa 6.2.3 Data Import -pyyntökäsittelijän yhteydessä.

Lokitiedon muoto

Käytännön esimerkeistä selviää myös, että lokitiedon muotoon kannattaa kiinnittää huomiota, jos tietomuotoon voi vaikuttaa. XML-tiedon käsittely on monimutkaisempaa kuin esimerkeissä käytetty CSV-tietomuoto. Tähän vaikuttaa osaltaan se, että XML-tiedossa yksittäinen rivi ei ole merkityksellinen vaan on käsiteltävä XML-elementtejä, kun taas CSV-tiedossa yksittäinen rivi on itsenäinen kokonaisuus, joka ei ole riippuvainen tiedoston muista riveistä. Tiedon käsittelyn monimutkaisuudella on vaikutusta myös indeksoinnin nopeuteen varsinkin, jos tiedostoissa on kymmeniätuhansia rivejä.

Indeksointitavat

Elasticsearch hakukoneeseen tieto vietiin esimerkeissä Logstash ohjelmalla, jonka käyttö perustuu suuressa määrin ohjelman konfiguraatiotiedostossa käytettäviin erilaisiin lisäosiin (plugins). Lisäosiin kannattaa tutustua ja harjoitella niiden käyttöä, jonka jälkeen ohjelman käyttö on sujuvampaa ja voi saadaa aikaan nopeasti toimivia indeksointi ratkaisuja. Lisäosia myös päivitetään jatkuvasti, niihin tulee uusia ominaisuuksia ja ne kehittyvät jatkuvasti, joten niiden päivittäminen voi olla tarpeellista tietyin väliajoin palvelimelle. Molemmat sekä XML- että CSV-tiedostot vietiin samaan tapaan Logstash ohjelmaa käyttämällä Elasticsearch hakukoneen indeksiin. Molemmille tietomuodoille tehtiin omat konfiguraatiotiedostot Logstash ohjelmalle ja niissä käytettiin tietomuodon tarvitsemia lisäosia käsiteltävän tietomuodon mukaan. XML- ja CSV-tiedostojen käsittely olisi voitu yhdistää myös samaan konfiguraatiotiedostoon, jolloin file-lisäosassa olisi tarkkailtu esimerkiksi CSV ja XML nimisiä kansioita ja sen jälkeen olisi käytetty erilaisia tyyppi tai lippu ratkaisuja filter-osion if-lausekkeissa.

Apache Solr hakukoneessa XML-tiedoston kanssa käytettiin erillistä XSLT-tiedostoa, joka sisälsi säännöt tiedon muuntamiseen Solrin vaatimaan XML-muotoon. XSLT-tiedosto otettiin käyttöön Solr hakukoneen Data Import -pyyntökäsittelijään (DIH) määritellyssä konfiguraatiotiedossa. Jos XML-tiedosto haluttaisiin eri muotoon niin riittää, että tarvittavat muutokset tehtäisiin ainoastaan XSLT-tiedostoon. XML-tiedostoa indeksoitaessa jouduttiin siis ensin tekemään muutoksia näihin neljään tiedostoon: data-config.xml (DIH konfiguraatiotiedosto), schema.xml, solrconfig.xml ja *omes3.xsl (XSLT-tiedosto),* ja muutosten jälkeen ZooKeeper ohjelmalla ladattiin muutetut konfiguraatiotiedostot takaisin indeksin käyttöön, koska käytössä oli SolrCloud versio. Näiden toimenpiteiden jälkeen tietoa voitiin viedä hakukoneeseen. CSV-tiedoston vieminen Solriin oli huomattavasti yksinkertaisempaa. Indeksin luonnin jälkeen riitti, että tietoa vietäessä latauskomennossa ilmoitettiin ladattava tietotyyppi CSV ja Solr osasi luoda CSV-tiedoston otsikkkorivin mukaiset kentät indeksiin. CSV-tiedoston kaikki rivit eivät kuitenkaan menneet indeksiin suoraan vaan tuli ongelma toisen rivin kohdalla johtuen yhden sarakkeen tietotyypin erosta verrattuna ensimmäisen rivin vastaavan sarakkeen sisältämään tietotyyppiin. Skeemaan tehtiin tarvittava muutos API-liitännän kautta ja tämän jälkeen CSV-tiedoston kaikki rivit menivät indeksiin. CSV-tiedoston kohdalla ei siis tehty muutoksia Solrin konfiguraatiotiedostoihin manuaalisesti ja tästä syystä myöskään ZooKeeper ohjelmaa ei tarvinnut käyttää.

Hadoop

Molempien hakukoneiden kohdalla CSV-esimerkissä indeksiin vietävä tiedosto sijaitsi Hadoopin HDFS-tiedostojärjestelmässä, josta se ensin kopioitiin Linux-palvelimelle, ja sen jälkeen indeksiin vienti tapahtui samaan tapaan kuin XML-tiedoston kohdalla. Käytännön toteutuksissa tämä ei välttämättä ole toimiva ratkaisu, vaan Hadoopia käytettäessä kannattaa miettiä myös muita indeksointiratkaisuja. Hakukoneisiin on olemassa Hadoop-liitäntöjä, kuten Elasticsearch for Hadoop ja HDFS for Solr. Näitä käyttämällä sekä indeksiin luetut tiedostot että itse indeksit voivat sijaita Hadoopissa. Tavalla, jolla tietoa viedään Hadoopin, voi myös olla vaikutusta käytettyihin ratkaisuihin. Yleensä Hadoopia käytetään tiedon pitempiaikaiseen varastointiin ja jos Hadoopiin vietävä tieto sijaitsee ensin jollain Linux- tai Windows-levyllä niin siinä tapauksessa tieto voitaisiin indeksoida esimerkeissä esitetyllä tavalla ennen tiedon siirtämämista Hadoopiin, jota on kuitenkin jossain vaiheessa tehtävä, jos tietoa halutaan säilyttää johtuen palvelimien rajallisesta tallennuskapasiteetista. Hadoopiin tallennattaessa saadaan samalla myös tietoturvaa Hadoopin tarjoaman vikasietoisuuden myötä. Tässä vaiheessa on myös hyvä mainita indekseihin liittyvänä asiana, että jos jostain syystä indeksi menetetään, eikä sitä ole varmistettu jollain tapaa, niin alkuperäiset tiedostot on oltava olemassa ja luettava uudelleen indeksiin, jos halutaan niiden sisältämää tietoa tutkia hakukoneella.

Indeksoinnin nopeus

Esimerkissä käytettyjen lokitiedostojen kanssa ei ollut suorituskykyyn liittyviä ongelmia, mutta käytännön toteutuksissa tähän on kiinnitettävä huomiota. Jos indeksiin vietäviä tiedostoja on satoja, joista kukin sisältää tuhansia rivejä, niin indeksointivaihe voi kestää muutamasta minuutista jopa tunteihin. Varsinkin visualisoinneissa, joissa halutaan kuvata jonkin järjestelmän reaaliaikaista tilaa, tällaisella viiveellä on merkitystä.

Ohjeistus

Vaikka Apache Solr on ollut kauemmin käytössä kuin Elasticsearch ja molemmille hakukoneille löytyy hyvät tukisivustot ja useita hyviä opaskirjoja niin työtä tehdessä Elasticsearch hakukoneelle löytyi enemmän käytännön esimerkkejä internetin sivustoilta. Tämä kertoo, että Elasticsearch hakukonetta käytetään tänä päivänä enemmän kuin Solr hakukonetta.

Johtopäätös

Elasticsearch – Logstash – Kibana -ohjelmilla on mahdollista saada aikaan suhteellisen nopeasti näyttäviä lähes reaaliaikaisia visualisointeja, joissa esimerkiksi pylväsdiagrammit muuttuvat dynaamisesti indeksiin luetun tiedon mukaan. Tämä voidaan saada aikaan Elasticin ohjelmia käyttämällä ilman Python tai Java-ohjelmointia. Jos ohjelmointikielet ovat hallinnassa, kannattaa tutustua hakukoneiden API-rajapintoihin, ja selvittää miten ohjelmointikielillä voidaan tietoa viedä indeksiin, koska silloin tiedon indeksointiin voidaan käyttää jo olemassa olevia ohjelmointitaitoja, eikä tarvitse opetella niin paljon hakukoneiden omia erityisiä indeksointitapoja.

8 POHDINTA

Opinnäytetyössä selvitettiin miten radioverkkoverkkojärjestelmien testausautomaation tuottamaa dataa voidaan analysoida avoimen lähdekoodin hakukoneilla. Opinnäytetyössä esitettiin käytännön esimerkit lokitiedon viemisestä hakukoneeseen, kun lokitieto on XML- ja CSV-tietomuotoa. Elasticsearch hakukoneeseen tieto vietiin Logstash nimisellä ohjelmalla, jonka jälkeen tietoa haettiin ja visualisoitiin Kibana ohjelmassa. Apache Solr hakukoneessa XML-tiedosto vietiin hakukoneeseen käyttämällä XSLT-kieltä ja Data Import-pyyntökäsittelijää ja tietoa visualisoitiin Banana ohjelmassa.

Opinnäytetyön perusteella Elasticsearch hakukone vaikuttaa toimivalta kokonaisuudelta, johon kuuluu Logstash ja Kibana ohjelmat (ELK stack). Kibana ohjelmalla on mahdollista tehdä suhteellisen vaivattomasti näyttäviä visualisointeja hakukoneeseen indeksoidusta tiedosta. Apache Solr hakukoneen yhteydessä ei tule visualisointi ohjelmaa, vaan työssä käytetty Banana ohjelma on asennettava erikseen. Molemmille hakukoneille löytyy hyvät dokumentaatiosivut internetistä, mutta Elasticsearch hakukoneen käyttäjäyhteisö on suurempi ja siitä syystä tietoa on saatavilla siihen liittyen runsaasti internetistä. Molemmilla hakukoneilla esimerkeissä käytetyt lokitiedot saatiin vietyä hakukoneeseen, mutta Logstash ohjelman käytön Elasticsearch hakukoneen kanssa oppi nopeammin kuin Apache Solr hakukoneen indeksointitavat.

Aiheen laajuudesta johtuen hakukoneista jäi monia selvitettäviä asioita. Alla on lueteltu muutamia selvitettäviä jatkokehittämiskohteita jaoteltuna otsikoiden alle.

Hakukoneet ja Hadoop

Tiedon määrän lisääntyessä myös tallennuskapasiteetin tarve kasvaa, joten erilaisia Hadoop-ratkaisuja otetaan enenevässä määrin käyttöön yrityksissä. Hakukoneiden käyttöä Hadoopin kanssa voisi tutkia lisää ja selvittää hakukoneiden Hadoop-liitäntöjä kuten ES (Elasticsearch) Hadoop, jolloin käytössä on Hadoop työkaluja, kuten Spark ja Pig. Myös Apache Solr hakukoneelle on tarjolla Hadoop-liitäntöjä, kuten HDFS for Solr, Pig for Solr ja Spark for Solr.

Hakukoneet ja koneoppiminen (ML, Machine Learning)

Myös koneoppimista pyritään lisäämään yrityksissä tänä päivänä voimakkaasti koko ajan ja ainakin Elasticsearch hakukoneen maksullisesta X-Pack -paketista löytyy Machine Learning -ominaisuus, jonka käyttöönotto voisi olla selvittämiskohde ja selvittää löytyykö Solr hakukoneesta vastaavaa.

Hakukoneiden tietoturva

Hakukoneiden tietoturva on ajankohtaista, kun halutaan rajoittaa ketkä voivat hakea hakukoneeseen tallennettua tietoa. Tässä kohden voi selvittää minkälaisia ratkaisuja hakukoneet tarjoavat tähän asiaan.

Hakukoneiden API selvitys

Hakukoneita on mahdollista käyttää yleisimmillä ohjelmointikielillä vietäessä tietoa indeksiin ja hakuja tehdessä. Ohjelmistorajapintojen käyttäminen voi vähentää hakukoneiden erityisominaisuuksien opiskelua ja siten nopeuttaa niiden käyttöönottoa.

Hakukoneiden indeksien hallinta

Hakukoneiden käyttöönotto tuo uusia asioita ylläpidettäviksi. Selvitettäviä asioita on indeksien ylläpito ja hallinta kuten niiden nimeäminen, vanhan tiedon poistaminen, tiedon varmistus ja palauttaminen ongelmatilanteissa ja toiminta niiden aikana, kun indeksit eivät ole käytettävissä.

Elasticsearch Beats-tuotteet

Elasticsearch hakukoneen Beats-tuotteet voivat olla käyttökelpoisia monessa tilanteessa. Opinnäytetyön Logstash-esimerkissä tiedosto vietiin indeksiin samalta palvelimelta, jossa ajettiin Logstash ohjelmaa ja Elasticsearch hakukonetta. Jos käytössä on useita palvelimia niin kannattaa tutustua Filebeat ohjelmaan, jolla voidaan lähettää tiedostoja Logstash ohjelman käsiteltäväksi.

Visualisointi

Kibana ohjelmassa on Timelion-visualisointityökalu, jota ei käsitelty opinnäytetyössä. Siinä on käytössä .es() niminen skriptikieli, jossa voi käyttää useita matemaattisia funktioita tiedon esittämiseen visuaalisesti. Selvittämiskohde voisi olla myös miten hakukoneissa tehtyjä visualisointeja voi hyödyntää upottamalla niitä omille räätälöidyille www-sivustoille.

LÄHTEET

Apache Hadoop 2017. WebHDFS REST API. Viitattu 3.4.2018, https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/WebHDFS.html.

Apache Lucene 2016. Apache Lucene Core. Viitattu 16.2.2018, https://lucene.apache.org/core/.

Apache Lucene 2018. Search Basics. Viitattu 16.2.2018, https://lucene.apache.org/core/7_2_1/core/org/apache/lucene/search/package-summary.html#package.description.

Apache Solr 2017a. A quick overview. Viitattu 2.3.2018, https://lucene.apache.org/solr/guide/7_2/a-quick-overview.html.

Apache Solr 2017b. Solr system requirements. Viitattu 2.3.2018, https://lucene.apache.org/solr/guide/7_2/solr-system-requirements.html.

Apache Solr 2017c. Running Solr. Viitattu 2.3.2018, https://lucene.apache.org/solr/guide/6_6/running-solr.html.

Apache Solr 2017d. Solr Cores and solr.xml. Viitattu 2.3.2018, https://lucene.apache.org/solr/guide/7_2/solr-cores-and-solr-xml.html.

Apache Solr 2017e. Solr Configuration Files. Viitattu 9.3.2018, https://lucene.apache.org/solr/guide/7_2/solr-configuration-files.html.

Apache Solr 2017f. Overview of Documents, Fields, and Schema Design. Viitattu 6.3.2018, https://lucene.apache.org/solr/guide/6_6/overview-of-documents-fields-and-schema-design.html.

Apache Solr 2017g. Schemaless Mode. Viitattu 6.3.2018, https://lucene.apache.org/solr/guide/6_6/schemaless-mode.html#schemaless-mode.

Apache Solr 2017h. Schema Factory Definition in SolrConfig. Viitattu 6.3.2018, https://lucene.apache.org/solr/guide/6_6/schema-factory-definition-in-solrconfig.html.

Apache Solr 2017i. Configuring solrconfig.xml. Viitattu 6.3.2018, https://lucene.apache.org/solr/guide/6_6/configuring-solrconfig-xml.html.

Apache Solr 2017j. Uploading Structured Data Store Data with the Data Import Handler. Viitattu 6.3.2018, https://lucene.apache.org/solr/guide/7_2/uploading-structured-data-store-data-with-the-data-import-handler.html.

Apache Solr 2017k. SolrCloud. Viitattu 7.3.2018, https://lucene.apache.org/solr/guide/7_2/solr-cloud.html.

Apache Solr 2017l. Getting Started with SolrCloud. Viitattu 8.3.2018, https://lucene.apache.org/solr/guide/7_2/getting-started-with-solrcloud.html#getting-started-with-solrcloud.

Apache Solr 2017m. Setting Up an External ZooKeeper Ensemble. Viitattu 8.3.2018, https://lucene.apache.org/solr/guide/7_2/setting-up-an-external-zookeeper-ensemble.html.

Apache Solr 2017n. Solr Control Script Reference. Viitattu 8.3.2018, https://lucene.apache.org/solr/guide/7_2/solr-control-script-reference.html#solr-control-script-reference.

Apache Solr 2017o. Uploading data with index handlers. Viitattu 12.3.2018, https://lucene.apache.org/solr/guide/7_2/uploading-data-with-index-handlers.html.

Apache Solr 2017p. Uploading structured data store data with the Data Import Handler. Viitattu 12.3.2018, https://lucene.apache.org/solr/guide/7_2/uploading-structured-data-store-data-with-the-data-import-handler.html#dih-concepts-and-terminology.

Apache Solr 2017q. Schema API. Viitattu 3.4.2018, https://lucene.apache.org/solr/guide/6_6/schema-api.html.

Apache Solr Wiki 2014a. SchemaXml. Viitattu 6.3.2018, https://wiki.apache.org/solr/SchemaXml.

Apache Solr Wiki 2014b. SolrCloud. Viitattu 7.3.2018, https://wiki.apache.org/solr/SolrCloud.

Apache ZooKeeper 2017. What is ZooKeeper? Viitattu 7.3.2018, http://zookeeper.apache.org/.

ComputerHope 2018a. Regex. Viitattu 13.2.2018, https://www.computerhope.com/jargon/r/regex.htm.

ComputerHope 2018b. Regex. Viitattu 13.2.2018, https://www.computerhope.com/unix/regex-quickref.htm.

Datatypic 2014. XSLT Functions. Viitattu 12.3.2018, http://www.xsltfunctions.com/.

Elastic 2018a. Basic Concepts. Viitattu 19.2.2018, https://www.elastic.co/guide/en/elasticsearch/reference/5.5/getting-started.html.

Elastic 2018b. Logstash. Viitattu 21.2.2018, https://www.elastic.co/products/logstash.

Elastic 2018c. Stashing Your First Event. Viitattu *21.2.2018,* https://www.elastic.co/guide/en/logstash/current/first-event.html.

Elastic 2018d. Configuring Logstash. Viitattu 21.2.2018, https://www.elastic.co/guide/en/logstash/current/configuration.html .

Elastic 2018e. Structure of a config file. Viitattu 21.2.2018, https://www.elastic.co/guide/en/logstash/current/configuration-file-structure.html.

Elastic 2018f. Codec plugins. Viitattu 21.2.2018, https://www.elastic.co/guide/en/logstash/current/codec-plugins.html.

Elastic 2018g. Accessing event data and fields in the configuration. Viitattu 22.2.2018, https://www.elastic.co/guide/en/logstash/current/event-dependent-configuration.html.

Elastic 2018h. Input plugins. Viitattu 22.2.2018, https://www.elastic.co/guide/en/logstash/current/input-plugins.html.

Elastic 2018i. Filter plugins. Viitattu 22.2.2018, https://www.elastic.co/guide/en/logstash/current/filter-plugins.html.

Elastic 2018j. Output plugins. Viitattu 22.2.2018, https://www.elastic.co/guide/en/logstash/current/output-plugins.html.

Elastic 2018k. Visualizing your data. Viitattu 23.3.2018, https://www.elastic.co/guide/en/kibana/current/tutorial-visualizing.html.

Fisher, T. 2018. Lifewire, What is a CSV file? Viitattu 9.2.2018, https://www.lifewire.com/csv-file-2622708.

Github 2018. Lucidworks/Banana. Viitattu 5.4.2018, https://github.com/Lucidworks/banana/.

Gupta, Y. & Gupta, R. K. 2017a. Mastering Elastic Stack, Packt Publishing. Viitattu 17.2.2018, http://proquest.safaribooksonline.com.ezp.oamk.fi:2048/book/databases/business-intelligence/9781786460011.

Homes, B. 2011. Fundamentals of Software Testing, John Wiley & Sons. Viitattu 1.2.2018, https://ebookcentral-proquest-com.ezp.oamk.fi:2047/lib/oamk-ebooks/detail.action?docID=1120766.

Hortonworks 2017. Webhdfs – http rest access to hdfs. Viitattu 3.4.2018, https://hortonworks.com/blog/webhdfs-http-rest-access-to-hdfs/.

IBM 2018. What is Big Data Analytics? Viitattu 27.3.2018, https://www.ibm.com/analytics/hadoop/big-data-analytics.

json.org 2018. Introducing JSON. Viitattu 9.2.2018, https://www.json.org/.

Karambelkar, H. V. 2014. Scaling Apache Solr, Packt Publishing. Viitattu 2.3.2018. http://ebookcentral.proquest.com/lib/oamk-ebooks/detail.action?docID=1706448.

Karambelkar, H.V. 2015. Scaling Big Data with Hadoop and Solr, Packt Publishing. Viitattu 6.3.2018, http://ebookcentral.proquest.com/lib/oamk-ebooks/detail.action?docID=2037702.

Lopez, F. & Romero V. 2014. Mastering Python Regular Expressions, Packt Publishing. Viitattu 10.2.2018, https://ebookcentral-proquest-com.ezp.oamk.fi:2047/lib/oamk-ebooks/detail.action?docID=1644026.

Paessler 2018. The Syslog Message Format. Viitattu 12.2.2018, https://www.paessler.com/it-explained/syslog.

Patil, D. 2015. Data Import Handler – import data from XML files which are in Solr xml format. Viitattu 12.3.2018, http://blog.thedigitalgroup.com/dattatrayap/data-import-handler-import-data-from-xml-files-which-are-in-solr-xml-format/.

Rogozinski, M. & Kuc, R. 2014. ElasticSearch Server 2nd edition, Packt Publishing. Viitattu 18.2.2018, https://ebookcentral-proquest-com.ezp.oamk.fi:2047/lib/oamk-ebooks/detail.action?docID=1674857.

Ruby community 2018, RubyGems. Viitattu 11.4.2018, https://rubygems.org.

Serafini, A 2013. Apache Solr Beginners Guide, Packt Publishing. Viitattu 26.3.2018, https://ebookcentral-proquest-com.ezp.oamk.fi:2047/lib/oamk-ebooks/detail.action?docID=1532006.

Sharma, V. 2016. Beginning Elastic Stack, Apress. Viitattu 23.2.2018, http://proquest.safaribooksonline.com.ezp.oamk.fi:2048/book/operating-systems-and-server-administration/9781484216941/firstchapter.

Smiley, D. 2014. Apache Solr Enterprise Search Server, Packt Publishing. Viitattu 9.3.2018, http://ebookcentral.proquest.com/lib/oamk-ebooks/detail.action?docID=2058654.

Splunk 2018. Logging overview. Viitattu 7.2.2018, http://dev.splunk.com/view/logging/SP-CAAAFCH.

Stackify 2017. Syslog Tutorial. Viitattu 11.2.2018, https://stackify.com/syslog-101/.

Turnbull, D. & Berryman, J. 2016. Relevant Search: With applications for Solr and Elasticsearch, Manning Publications. Viitattu 14.2.2018, http://proquest.safaribook-sonline.com.ezp.oamk.fi:2048/book/web-development/search/9781617292774.

Tutorialspoint 2018a. XML – Overview. Viitattu 8.2.2018, https://www.tutorialspoint.com/xml/xml_overview.htm.

Tutorialspoint 2018b. XML – Syntax. Viitattu 8.2.2018, https://www.tutorialspoint.com/xml/xml_syntax.htm.

Tutorialspoint 2018c. XML – Schemas. Viitattu 8.2.2018, https://www.tutorialspoint.com/xml/xml_schemas.htm.

Tutorialspoint 2018d. XML – Overview. Viitattu 8.2.2018, https://www.tutorialspoint.com/xml/xslt_overview.htm.

Tutorialspoint 2018e. Lucene – Overview. Viitattu 16.2.2018, https://www.tutorialspoint.com/lucene/lucene_overview.htm.

Tutorialspoint 2018f. Hadoop - Introduction. Viitattu 27.3.2018, https://www.tutorialspoint.com/hadoop/hadoop_introduction.htm.

Tutorialspoint 2018g. Hadoop – HDFS Overview. Viitattu 27.3.2018, https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm.

Tutorialspoint 2018h. Hadoop – HDFS Operations. Viitattu 27.3.2018, https://www.tutorialspoint.com/hadoop/hadoop_hdfs_operations.htm.

Tutorialspoint 2018i. curl – Unix, Linux Command. Viitattu 3.4.2018, https://www.tutorialspoint.com/hadoop/ unix_commands/curl.htm.

w3schools 2018a. XML DOM Tutorial. Viitattu 9.2.2018, https://www.w3schools.com/xml/dom_nodes.asp.

w3schools 2018b. XPath Tutorial. Viitattu 9.2.2018, https://www.w3schools.com/xml/xpath_intro.asp.

w3schools 2018c. XPath Syntax. Viitattu 9.2.2018, https://www.w3schools.com/xml/xpath_syntax.asp.

w3schools 2018d. XSLT Introduction. Viitattu 9.2.2018, https://www.w3schools.com/xml/xsl_intro.asp.

w3schools 2018e. JSON Syntax. Viitattu 9.2.2018, https://www.w3schools.com/js/js_json_syntax.asp.

Xmlper 2018. XSLT Transform. Viitattu 11.4.2018, http://www.xmlper.com/.

SYSLOG VIESTIN PRIORITY-OSAN FACILITY- JA SEVERITY-ARVOT LIITE 1

Facility-arvot (Paessler 2018, viitattu 12.2.2018)

Arvo	Facility kuvaus
0	Kernel messages
1	User-level messages
2	Mail System
3	System Daemons
4	Security / Authorization Messages
5	Messages generated by syslogd
6	Line Printer Subsystem
7	Network News Subsystem
8	UUCP Subsystem
9	Clock Daemon
10	Security / Authorization Messages
11	FTP Daemon
12	NTP Subsystem
13	Log Audit
14	Log Alert
15	Clock Daemon
16 – 23	Local Use 0 – 7

Severity-arvot (Paessler 2018, viitattu 12.2.2018)

Koodi	Severity	Kuvaus
0	Emergency	System is unusable
1	Alert	Action must be taken immediately
2	Critical	Critical conditions
3	Error	Error conditions
4	Warning	Warning conditions
5	Notice	Normal but significant condition
6	Informational	Informational messages
7	Debug	Debug-level messages

SÄÄNNÖLLISEN LAUSEKKEEN ERIKOISMERKKEJÄ LIITE 2

Säännöllisen lausekkeen erikoismerkkien kuvaukset (ComputerHope 2018a, viitattu 13.2.2018)

Erikois-merkki	Kuvaus	Regex lauseke	Merkkijonon vastaavuus
^	Vastaa merkkijonon alkua, kun käytetään hakasulkujen sisällä niin kääntää merkityksen	^abc	abc, abcdef, abc123…
$	Erikoismerkillä määritetään haettavann merkkijonon loppu	abc$	my:abc, 123abc, theabc…
.	Piste edustaa mitä tahansa yhtä merkkiä	x.z	xbz, xcz…
[...]	Sisältää yksittäisiä merkkejä tai arvo alueita, joiden pitää täsmätä merkkijonon kanssa	[e4K]	e,4 tai K
[^...]	Kääntää edellisen eli täsmää merkkien kanssa, jotka eivät ole hakasulkujen sisällä	[^e4K]	xay, 123, Rxs…
[a-z]	Merkkijonossa voi olla mitkä tahansa merkit väliltä 'a' ja 'z'	[b-Z]	bc, mind, xyz…
{x}	Määritettävän merkkijonon tulee esiintyä tarkalleen x -kertaa	(abc){2}	abcabc, abcabcCC…
{x,}	Määritettävän merkkijonon tulee esiintyä x -kertaa tai useammin	(abc){2,}	abcabc, abcabcabc…
{x,y}	Määritetty merkkijono voi esiintyä x -y kertaa	(a){2,4}	aa, aaa, aaaa
?	Erikoimerkkiä edeltävä merkki voi esiintyä merkkijonossa 0-1 kertaa	ab?c	ac, abc
+	Erikoismerkkiä edeltävät merkit tai arvoalueet voivat täsmätä yhden tai useamman kerran	a+c	ac, aac, aaac…
*	Erikoismerkkiä välittömästi edeltävä merkki voi täsmätä monta kertaa tai ei kertaakaan	ab*c	abc, abbcc, abcdc…
(...)	Merkkijonossa oltava mitä laitettu sulkujen sisälle	(kis)	kisailu, kissa, kisa…
|	Erikoismerkillä määritetään täsmättävän merkkijonon vaihtoehtoinen kirjoitusasu, on siis TAI -operaattori	abc|xyz	abc tai xyz

REGEX LYHENTEITÄ LIITE 3

Regex lyhenteitä (ComputerHope 2018a, viitattu 13.2.2018)

Lyhenne	Kuvaus	Regex lauseke	Merkkijonon vastaavuus
\s	Vastaa välilyöntiä, tabulaattoria	abcd\se	abcd e, abcd(tab)e
\S	Vastaa ei tyhjää merkkiä: välilyönti, tabulaattori	\S\S\s\S	AB D, RT k, 99(tab)9
\w	Mikä tahansa merkki: kirjain, numero tai alaviiva, vastaa Regex lauseketta: [a-zA-Z0-9_]	\w+\s\s\w+	2323 232, re 4343…
\W	Edellä olevan vastakohta, mikä tahansa muu merkki kuin kirjain, numero tai alaviiva, vastaava Regex lauseke: [^a-zA-Z0-9_]	\w+\s\s\w+	%% ((, ## ??
\d	Vastaa mitä tahansa numeroa, vastaava Regex lauseke: [0-9]	\d{4}\s	4432, 4789…
\D	Edellisen vastakohta, mikä tahansa ei -numero	\D{4}\s	aaaa, aert, xxxx…